aws

开发人员指南

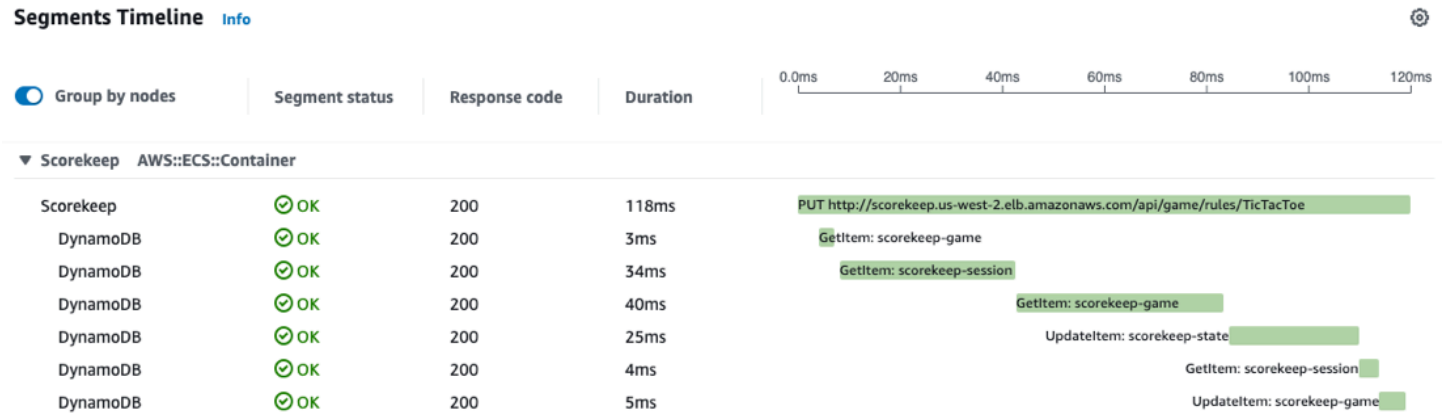
AWS X-Ray

AWS X-Ray: 开发人员指南

Table of Contents

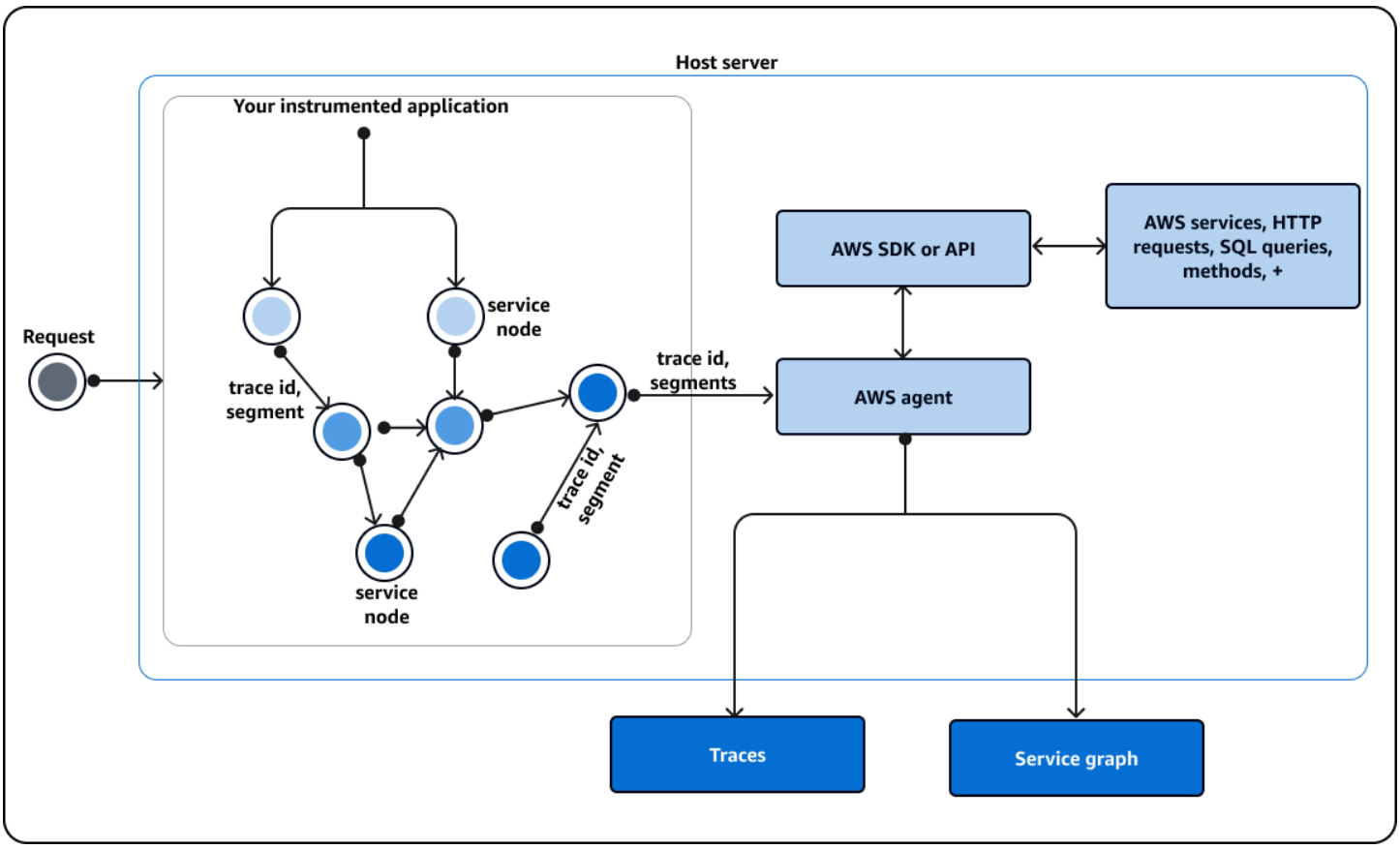
什么是 AWS X-Ray？

AWS X-Ray 是一项服务，它收集有关您的应用程序所处理的请求的数据，并提供可用于查看、筛选和深入了解这些数据的工具，以识别问题和优化机会。对于对应用程序的任何跟踪请求，您不仅可以查看有关请求和响应的详细信息，还可以查看有关应用程序对下游 AWS 资源、微服务、数据库和 Web APIs 的调用的详细信息。



AWS X-Ray 除了已与 X-Ray 集成的应用程序用途外，还会接收来自 AWS 服务 您的应用程序的跟踪。检测应用程序涉及发送应用程序内传入和出站请求及其他事件的跟踪数据，以及与每个请求相关的元数据。许多检测场景只需要配置更改。例如，您可以检测您的 Java 应用程序发出的所有传入 HTTP 请求和下游调用。 AWS 服务 有几个 SDKs、代理和工具可用于检测您的应用程序，以进行 X-Ray 跟踪。有关更多信息，请参阅检测应用程序。

AWS 服务 与 X-Ray 集成的可以向传入的请求添加跟踪标头、向 X-Ray 发送跟踪数据或运行 X-Ray 守护程序。例如， AWS Lambda 可以将有关请求的跟踪数据发送到您的 Lambda 函数，并在工作程序上运行 X-Ray 守护程序，以便更轻松地使用 X-Ray SDK。



每个客户端 SDK 不是直接将跟踪数据发送到 X-Ray，而是将 JSON 分段文档发送到侦听 UDP 流量的进程守护程序进程。X-Ray 进程守护程序将分段缓冲在队列中，并将分段批量上传到 X-Ray。该守护程序可用于 Linux、Windows 和 macOS，并包含在和平台 AWS Elastic Beanstalk 上 AWS Lambda 。

X-Ray 使用来自支持云应用程序的 AWS 资源的跟踪数据来生成详细的跟踪地图。该跟踪地图显示客户端、您的前端服务以及前端服务调用来处理请求和保存数据的后端服务。您可以使用跟踪地图来查明瓶颈、延迟峰值和其他需要解决的问题，以提高应用程序性能。


scorekeep-move
DynamoDB Table
Client
Scorekeep
ECS Container
SNS
SNS
scorekeep-session
DynamoDB Table

X-Ray 入门

要使用 X-Ray，请执行以下步骤：

1. 检测您的应用程序，以允许 X-Ray 跟踪您的应用程序如何处理请求。

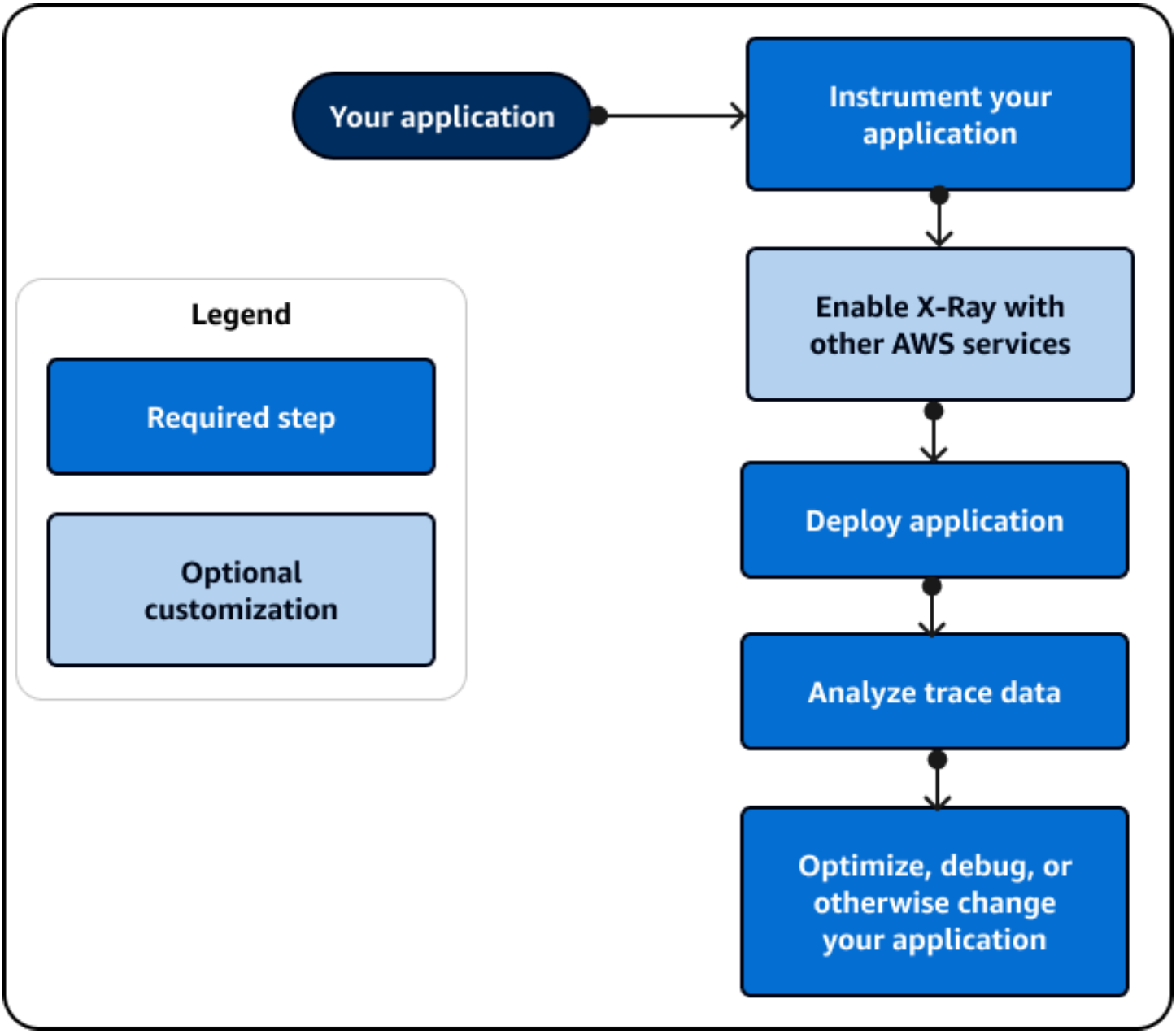
   - 使用 X-Ray SDKs APIs、X-Ray ADOT 或 CloudWatch 应用程序信号，用于向 X-Ray 发送跟踪数据。有关要使用哪个接口的更多信息，请参阅选择界面。

   有关检测的更多信息，请参阅正在对您的应用程序进行检测 AWS X-Ray。

2. （可选）将 X-Ray 配置为与其他 AWS 服务 与 X-Ray 集成的设备一起使用。您可以对跟踪采样并将标头添加到传入请求，运行代理或收集器，然后自动将跟踪数据发送到 X-Ray。有关更多信息，请参阅 AWS X-Ray 与其他人集成 AWS 服务。
3. 部署您检测到的应用程序。当您的应用程序收到请求时，X-Ray SDK 将记录跟踪、分段和子分段数据。在此步骤中，您可能还必须设置 IAM 策略并部署代理或收集器。

   - 有关使用 D AWS istro for OpenTelemetry (ADOT) SDK 和 CloudWatch 代理在不同平台上部署应用程序的脚本示例，请参阅应用程序信号演示脚本。
   - 有关使用 X-Ray SDK 和 X-Ray 进程守护程序部署应用程序的脚本示例，请参阅 AWS X-Ray 示例应用程序。
4. （可选）打开控制台以查看和分析数据。您可以查看跟踪地图、服务地图等的 GUI 表示形式，以检查应用程序的运行情况。使用控制台中的图形信息来优化、调试和了解您的应用程序。有关选择控制台的更多信息，请参阅使用控制台。

下图展示了如何开始使用 X-Ray：



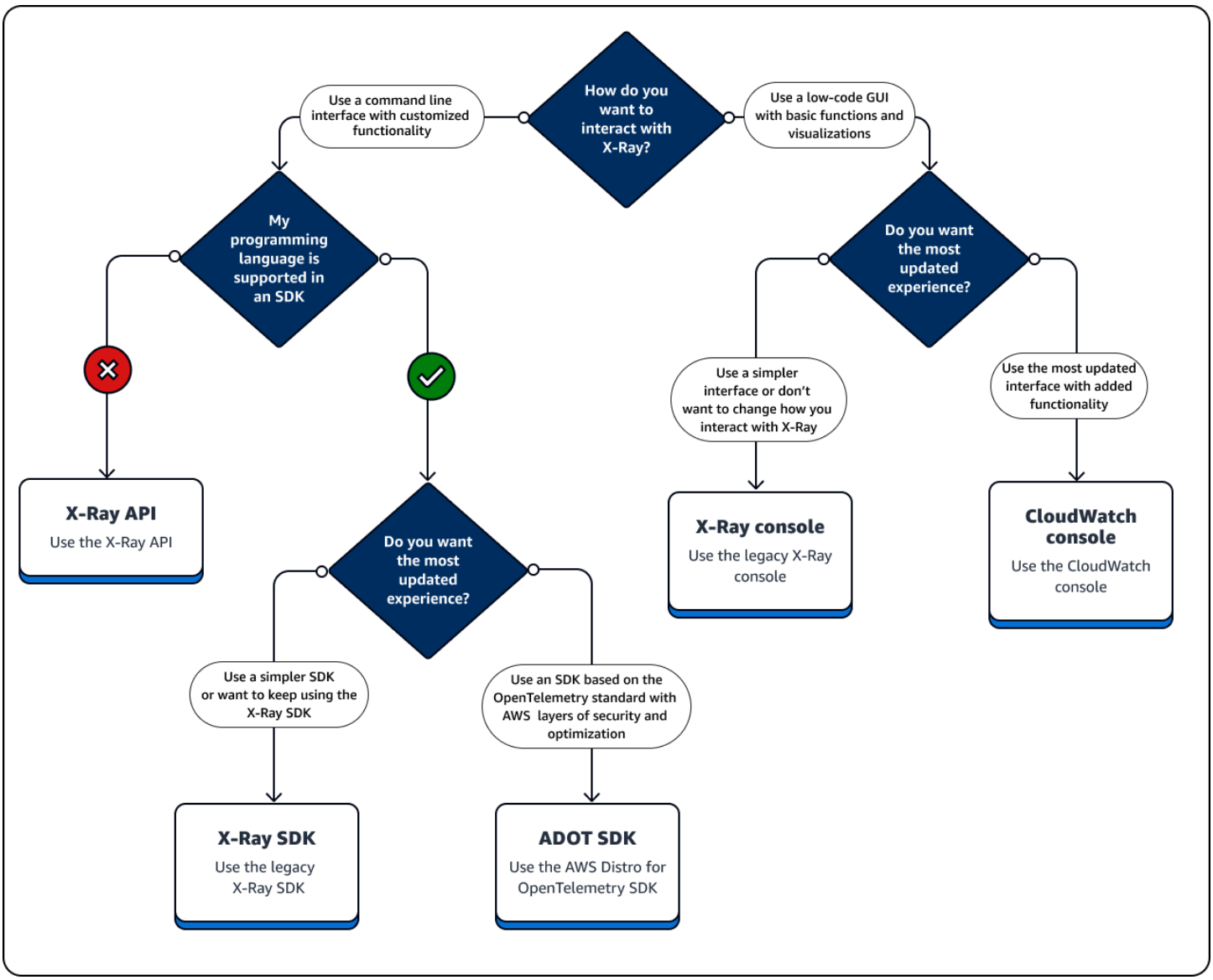
有关控制台中提供的数据和地图的示例，请启动已经过检测，可生成跟踪数据的应用程序示例。在几分钟内，您可以生成流量，将分段发送到 X-Ray，并查看跟踪和服务地图。

选择界面

AWS X-Ray 可以深入了解您的应用程序的工作原理以及它与其他服务和资源的交互情况。在对应用程序进行检测或配置后，X-Ray 会在您的应用程序处理请求时收集跟踪数据。您可以分析这些跟踪数据，以识别性能问题、排查错误并优化资源。本指南向您展示如何使用以下指南与 X-Ray 进行交互：

- AWS Management Console 如果您想快速入门，或者可以使用预先构建的可视化来执行基本任务，请使用。
  - 选择 Amazon CloudWatch 控制台，获取包含 X-Ray 控制台所有功能的最新用户体验。
  - 如果您想要更简单的界面或不想更改与 X-Ray 的交互方式，请使用 X-Ray 控制台。
- 如果您需要的自定义跟踪、监控或日志记录功能超出了其所 AWS Management Console 能提供的范围，请使用 SDK。
  - 选择 ADOT 如果你想要一个基于开源的不受供应商限制的 SDK，请使用 SDK OpenTelemetry SDK 增加了 AWS 安全层和优化层。
  - 如果您想要更简单的 SDK 或不想更新应用程序代码，请选择 X-Ray SDK。
- 如果 SDK 不支持您的应用程序的编程语言，请使用 X-Ray API 操作。

下图可帮助您选择如何与 X-Ray 进行交互：



了解界面类型

- 使用 SDK
- 使用控制台
- 使用 X-Ray API

使用 SDK

如果您想使用命令行界面，或者需要的自定义跟踪、监控或日志记录功能超出 AWS Management Console中提供的范围，请使用 SDK。您也可以使用 S AWS DK 开发使用 X-Ray 的程序 APIs。你可以使用 AWS Distro for OpenTelemetry (ADOT) SDK 或 X-Ray SDK。

如果您使用 SDK，则可以在检测应用程序和配置收集器或代理时为工作流添加自定义。您可以使用 SDK 来执行以下无法使用 AWS Management Console完成的任务：

- 发布自定义指标 - 以低至 1 秒的高分辨率对指标采样，使用多个维度添加有关指标的信息，并将数据点聚合到统计数据集中。
- 自定义收集器 - 自定义收集器任何部分的配置，包括接收器、处理器、导出器和连接器。
- 自定义您的检测 - 自定义分段和子分段，将自定义键值对添加为属性，并创建自定义指标。
- 以编程方式创建和更新采样规则。

使用 ADOT 如果你想灵活地使用标准化的 SDK OpenTelemetry SDK 增加了 AWS 安全层和优化层。的 AWS 发行版 (OpenTelemetry ADOT) SDK 是一个与供应商无关的软件包，它允许与其他供应商和非AWS 服务的后端集成，而无需重新分析您的代码。

如果您已经在使用 X-Ray SDK，只与 AWS 后端集成，并且不想更改与 X-Ray 或应用程序代码的交互方式，请使用 X-Ray SDK。

有关每项特征的更多信息，请参阅在 AWS Distro for 和 X-Ray OpenTelemetry 之间进行选择 SDKs。

使用 ADOT SDK

这些区域有：ADOT SDK 是一组向后端服务发送数据的开源 APIs、库和代理。ADOT 由多个后端和代理支持 AWS，与多个后端和代理集成，并提供大量由后端和代理维护的开源库 OpenTelemetry 社区。使用 ADOT SDK 用于检测您的应用程序并收集日志、元数据、指标和跟踪。你也可以使用 ADOT 监控服务并根据中的指标设置警报 CloudWatch。

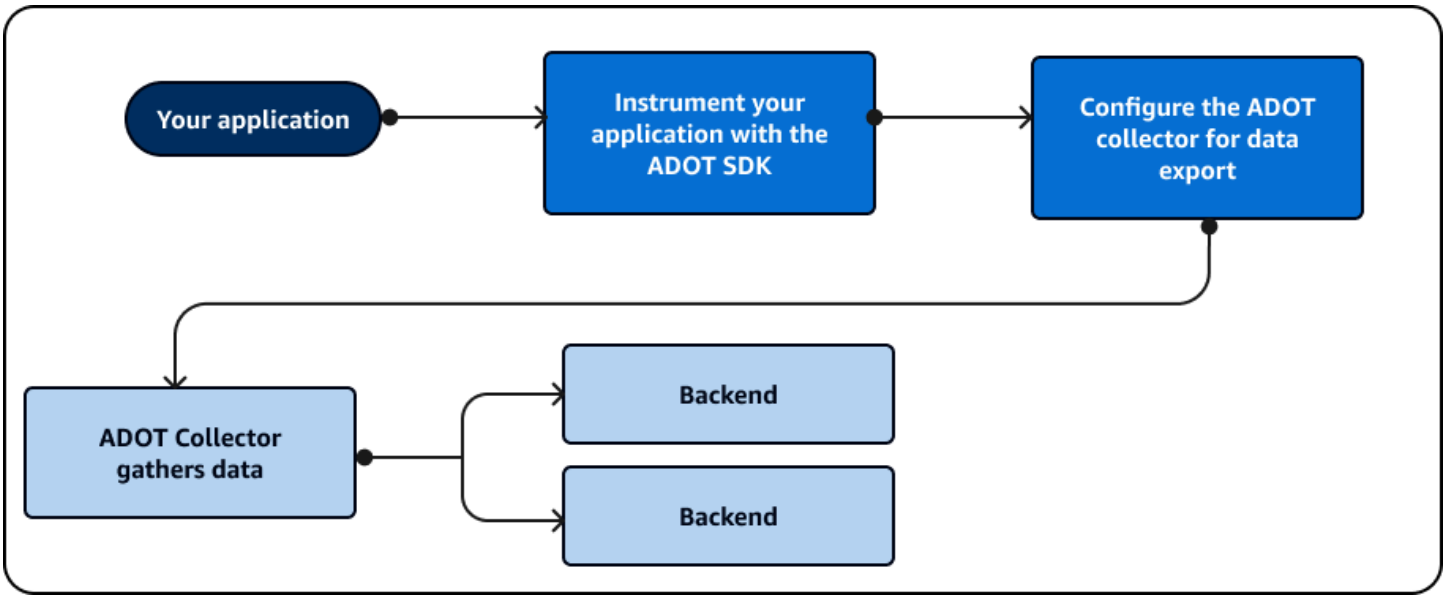
如果你正在使用 ADOT SDK，你有以下选项，再加上代理：

- 使用 ADOT 使用CloudWatch 代理进行 SDK — 推荐。
- 使用 ADOT 带有 SDK ADOT Collector — 如果您想使用具有多 AWS 层安全性和优化的独立于供应商的软件，则建议您使用。

要再次使用 ADOT SDK，请执行以下操作：

- 使用对应用程序进行仪器测试 ADOT SDK。有关更多信息，请参阅 ADOT 技术文档中适用于编程语言的文档。
- 配置一个 ADOT 收集器告诉它要将收集的数据发送到哪里。

之后 ADOT collector 接收您的数据，然后将其发送到您在中指定的后端 ADOT 配置。ADOT 可以将数据发送到多个后端，包括外部的供应商 AWS，如下图所示：



AWS 定期更新 ADOT 以添加功能并与OpenTelemetry框架保持一致。更新和 future 开发计划 ADOT 是向公众开放的路线图的一部分。ADOT 支持多种编程语言，其中包括：

- Go
- Java
- JavaScript
- Python
- .NET
- Ruby
- PHP

如果你使用的是 Python，ADOT 可以自动检测您的应用程序。要开始使用 ADOT，请参阅 Collect o OpenTelemetry r AWS 发行版简介和入门。

使用 X-Ray SDK

X-Ray SDK 是一组向 AWS 后端服务发送数据的 AWS APIs 和库。使用 X-Ray SDK 可检测您的应用程序并收集跟踪数据。您无法使用 X-Ray SDK 收集日志或指标数据。

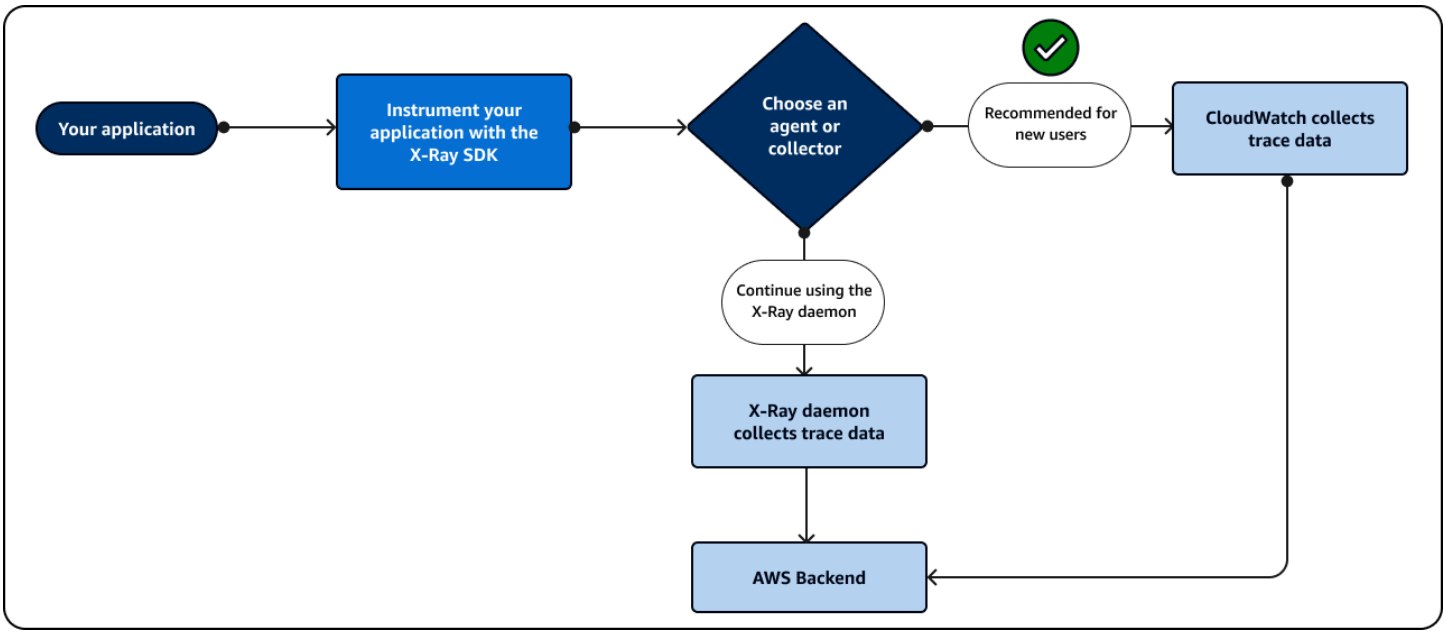
如果您使用的是 X-Ray SDK，则可以将以下选项与代理结合使用：

- 结合使用 X-Ray SDK 和 AWS X-Ray 守护程序 - 如果您不想更新应用程序代码，请使用此选项。
- 将 X-Ray SDK 与 CloudWatch 代理一起使用 —（推荐）CloudWatch 代理与 X-Ray SDK 兼容。

要使用 X-Ray SDK，请执行以下操作：

- 使用 X-Ray SDK 检测您的应用程序。
- 配置收集器以告知其将收集到的数据发送到何处。您可以使用 CloudWatch 代理或 X-Ray 守护程序来收集您的跟踪信息。

收集器或代理收到您的数据后，它会将其发送到您在代理配置中指定的 AWS 后端。X-Ray SDK 只能向 AWS 后端发送数据，如下图所示：



如果你正在使用 Java，您可以使用 X-Ray SDK 自动检测您的应用程序。要开始使用 X-Ray SDK，请查看与以下编程语言相关的库：

- Go
- Java
- Node.js
- Python
- .NET
- Ruby

使用控制台

如果您想要使用只需最少编码的图形用户界面（GUI），请使用控制台。不熟悉 X-Ray 的用户可以使用预先构建的可视化效果快速入门，并执行基本任务。您可以直接从控制台执行以下操作：

- 启用 X-Ray。
- 查看应用程序性能的简单摘要。
- 检查应用程序的运行状况。
- 识别高级别错误。
- 查看基本跟踪摘要。

你可以使用位于的亚马逊 CloudWatch 主机https://console.aws.amazon.com/cloudwatch/或https://console.aws.amazon.com/xray/家里的 X-Ray 主机与 X-Ray 进行交互。

使用亚马逊 CloudWatch 控制台

该 CloudWatch 控制台包括新的 X-Ray 功能，这些功能是从 X-Ray 控制台重新设计的，使其更易于使用。如果您使用 CloudWatch 控制台，则可以查看 CloudWatch 日志和指标以及 X-Ray 跟踪数据。使用 CloudWatch 控制台查看和分析数据，包括以下内容：

- X-Ray 跟踪 - 在应用程序处理请求时查看、分析和筛选与其关联的跟踪。使用这些跟踪可查找高延迟、调试错误并优化您的应用程序工作流。查看跟踪地图和服务地图，以查看应用程序工作流的可视化形式。
- 日志 - 查看、分析和筛选应用程序生成的日志。使用日志可排查错误，并根据特定的日志值设置监控。
- 指标 - 使用您的资源发出的指标或创建您自己的指标，来衡量和监控您的应用程序性能。以图形和图表的形式来查看这些指标。
- 监控网络和基础设施-监控主要网络的中断情况以及基础设施（包括容器化应用程序、其他 AWS 服务和客户端）的运行状况和性能。
- 下面的使用 X-Ray 控制台一节列出了 X-Ray 控制台中的所有功能。

有关 CloudWatch 控制台的更多信息，请参阅 Amazon 入门 CloudWatch。

登录亚马逊 CloudWatch 控制台，网址为https://console.aws.amazon.com/cloudwatch/。

使用 X-Ray 控制台

X-Ray 控制台为应用程序请求提供分布式跟踪。如果您想要更简单的控制台体验或不想更新应用程序代码，请使用 X-Ray 控制台。 AWS 不再开发 X-Ray 控制台。X-Ray 控制台包含以下用于检测应用程序的特征：

- 见解 - 自动检测应用程序性能中的异常并找出根本原因。见解包含在 CloudWatch 控制台的 Ins ights 下。有关更多信息，请参阅使用 X-Ray 控制台中的使用 X-Ray Insights。
- 服务地图 - 查看应用程序的图形结构及其与客户端、资源、服务和依赖项的连接。
- 跟踪 - 查看应用程序在处理请求时生成的跟踪的概述。使用跟踪数据来了解您的应用程序在基本指标方面的表现，包括 HTTP 响应和响应时间。
- 分析 - 使用图表解释、浏览和分析跟踪数据，以了解响应时间分布。
- 配置 - 创建自定义跟踪以更改以下各项的默认配置：
  - 采样 - 创建规则，以定义对应用程序采样以获取跟踪信息的频率。有关更多信息，请参阅使用 X-Ray 控制台中的配置采样规则。
  - 加密 - 使用密钥对静态数据加密，您可以使用 AWS Key Management Service对该密钥进行审核或禁用。
  - 分组 - 使用筛选条件表达式来定义一组具有共同特征（例如 URL 名称或响应时间）的跟踪。有关更多信息，请参阅配置分组。

在https://console.aws.amazon.com/xray/家中登录 X-Ray 控制台。

深入了解 X-Ray 控制台

使用 X-Ray 控制台可查看您的应用程序提供服务的请求的服务和关联跟踪的地图，并配置影响将跟踪发送到 X-Ray 的方式的分组和采样规则。

> **Note**
>
> X-Ray 服务地图和 CloudWatch ServiceLens 地图已合并到亚马逊 CloudWatch 控制台中的 X-Ray 追踪地图中。打开CloudWatch控制台，然后在左侧导航窗格的 X-Ray 轨迹下选择 Trace Map。
>
> CloudWatch 现在包括应用程序信号，它可以发现和监控您的应用程序服务、客户端、Synthetics 金丝雀和服务依赖关系。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查看运行状况指标（SLOs），并深入查看相关的 X-Ray 跟踪以获取更详细的故障排除。

主 X-Ray 控制台页面是跟踪地图，是 JSON 服务图的可视化形式，由 X-Ray 从您的应用程序生成的跟踪数据生成。该地图包含您账户中为请求提供服务的每个应用程序的服务节点，表示请求来源的上游客户端节点以及表示应用程序在处理请求时使用的 Web 服务和资源的下游服务节点。此外，还提供其他页面来查看跟踪和跟踪详情，以及配置组和采样规则。

在以下各节中查看 X-Ray 的主机体验并与 CloudWatch 主机进行比较。

探索 X-Ray 和 CloudWatch 游戏机

- 使用 X-Ray 跟踪地图
- 查看跟踪和跟踪详情
- 使用筛选条件表达式
- 跨账户跟踪
- 跟踪事件驱动型应用程序
- 使用延迟直方图
- 使用 X-Ray 见解
- 与 Analytics 控制台交互
- 配置组
- 配置采样规则
- 控制台深层链接

使用 X-Ray 跟踪地图

查看 X-Ray 跟踪地图来识别出现错误的服务、具有高延迟的连接或针对不成功请求的跟踪。

> **Note**
>
> CloudWatch 现在包括 A pplication Sig nals，它可以发现和监控您的应用程序服务、客户端、合成金丝雀和服务依赖关系。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查看运行状况指标（SLOs），并深入查看相关的 X-Ray 跟踪以获取更详细的故障排除。
>
> 在亚马逊 CloudWatch控制台中，X-Ray 服务 CloudWatch ServiceLens 地图和地图合并为一个 X-Ray 跟踪地图。打开CloudWatch 控制台，然后在左侧导航窗格的 X-Ray 轨迹下选择 Trace Map。

查看跟踪映射

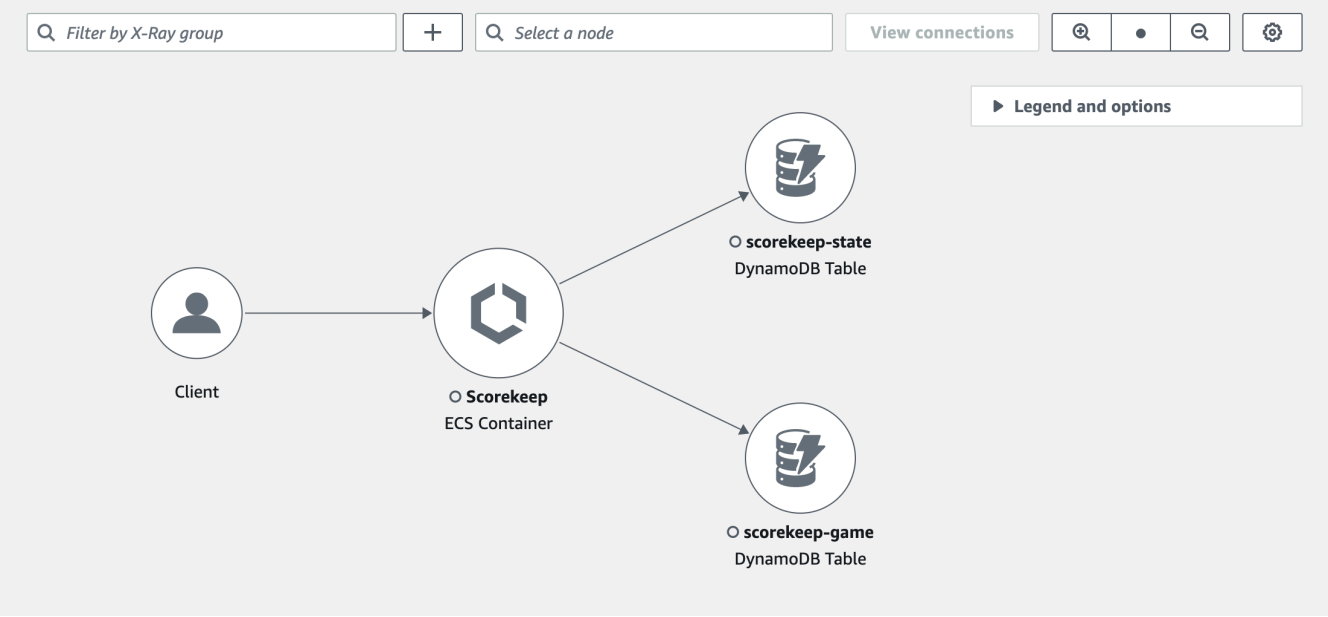
跟踪地图是跟踪数据的可视化形式，此类数据由您的应用程序生成。地图显示为请求提供服务的服务节点，表示请求来源的上游客户端节点以及表示应用程序在处理请求时使用的 Web 服务和资源的下游服务节点。

跟踪地图显示使用 Amazon SQS 和 Lambda 的不同事件驱动型应用程序中跟踪的互联视图。有关更多信息，请参阅跟踪事件驱动型应用程序。跟踪地图还支持跨账户跟踪，在一张地图中显示多个账户中的节点。

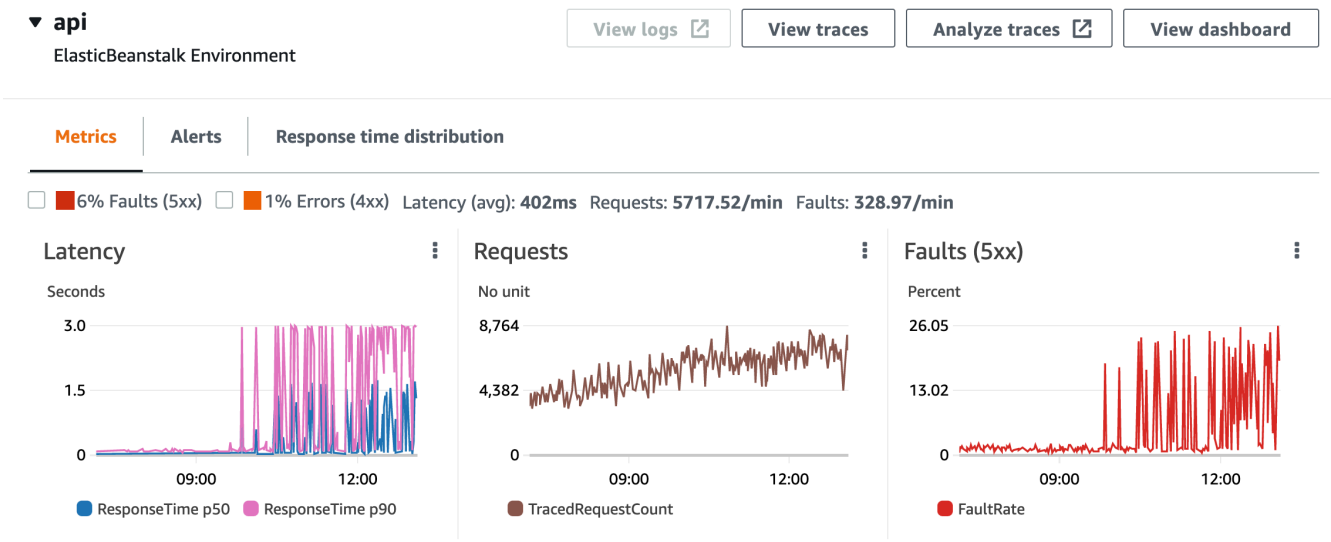
CloudWatch console

在 CloudWatch 控制台中查看追踪地图

1. 打开 CloudWatch 管理控制台。在左侧导航窗格的 X-Ray 跟踪部分下选择跟踪地图。



2. 选择一个服务节点来查看该节点的请求，或选择两个节点之间的边缘来查看经过该连接的请求。
3. 其他信息显示在跟踪地图下方，其中包括指标、警报以及响应时间分布的选项卡。在指标选项卡上，选择每张图的范围以深入查看更多详情，或选择故障或错误选项以筛选跟踪。在呼应时间分布选项卡上，选择在图内的一个范围以按照呼应时间来筛选跟踪。



4. 选择查看跟踪查看跟踪，或者如果已应用筛选条件，请选择查看经过筛选的跟踪。
5. 选择 “查看日志” 以查看与所选节点关联的 CloudWatch 日志。并非所有跟踪地图节点都支持查看日志。有关更多信息，请参阅故障排除 CloudWatch 日志。

跟踪地图通过用颜色概述每个节点来表示每个节点存在的问题：

- 红色表示服务器故障（500 系列错误）
- 黄色表示客户端错误（400 系列错误）
- 紫色表示限制错误（429 请求过多）

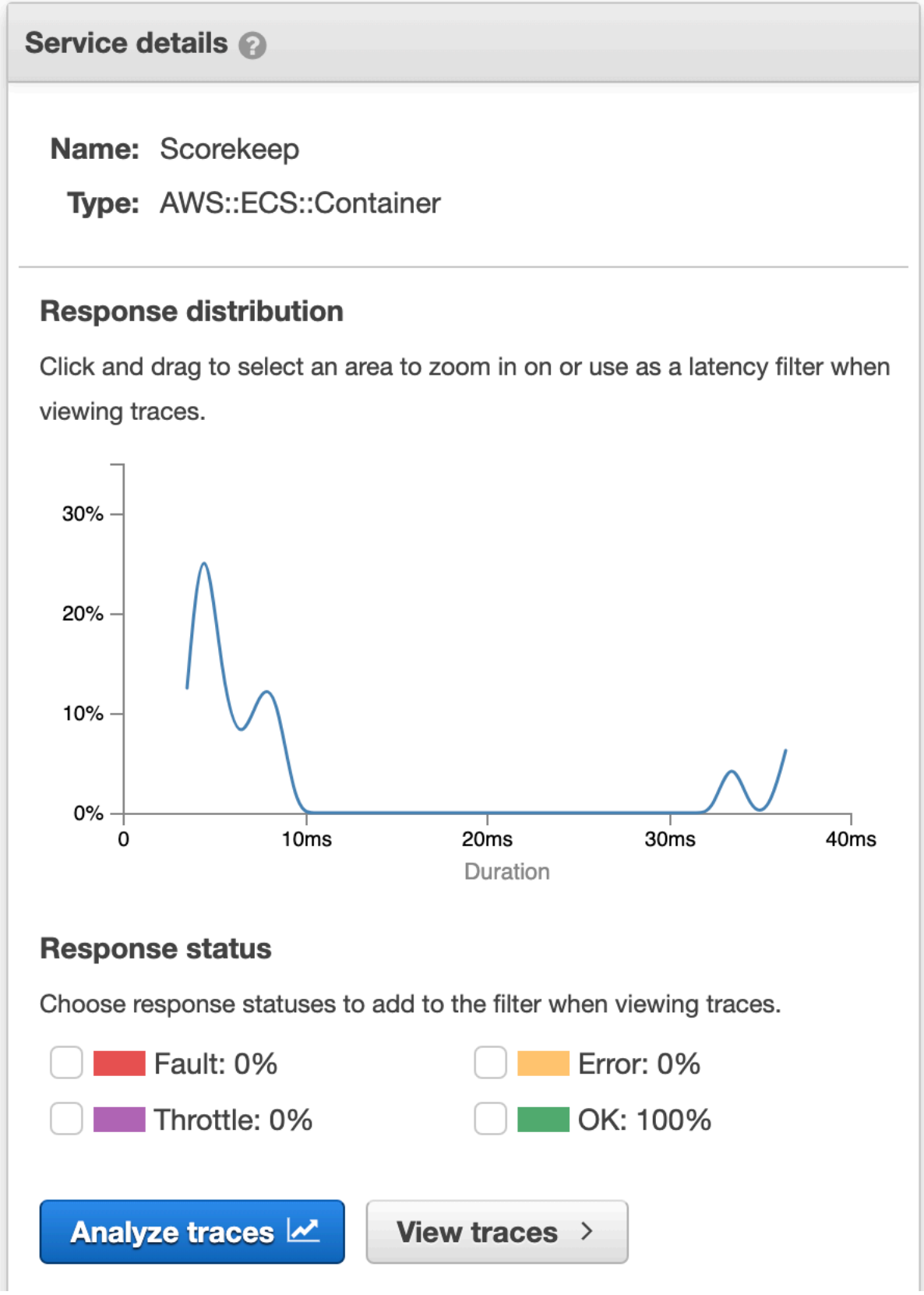
如果您的跟踪地图较大，请使用屏幕上的控件或鼠标放大、缩小和移动地图。

X-Ray console

查看服务地图

1. 打开 X-Ray 控制台。默认情况下，将显示服务地图。也可以从左侧导航窗格中选择服务地图。

2. 选择一个服务节点来查看该节点的请求，或选择两个节点之间的边缘来查看经过该连接的请求。
3. 使用呼应分布直方图按持续时间筛选跟踪，并选择要查看其跟踪的状态代码。然后选择查看跟踪打开应用筛选条件表达式后的跟踪列表。


Service details
Name: Scorekeep
Type: AWS::ECS::Container
Response distribution
Click and drag to select an area to zoom in on or use as a latency filter when viewing traces.
30%
20%
10%
0%
0
10ms
20ms
30ms
40ms
Duration
Response status
Choose response statuses to add to the filter when viewing traces.
Fault: 0%
Error: 0%
Throttle: 0%
OK: 100%
Analyze traces
View traces

服务地图根据成功调用与错误和故障的比率为每个节点显示颜色，从而指示节点的运行状况：

- 绿色表示成功调用
- 红色表示服务器故障（500 系列错误）
- 黄色表示客户端错误（400 系列错误）
- 紫色表示限制错误（429 请求过多）

如果您的服务地图较大，则使用屏幕上的控件或鼠标可放大、缩小和移动该图像。

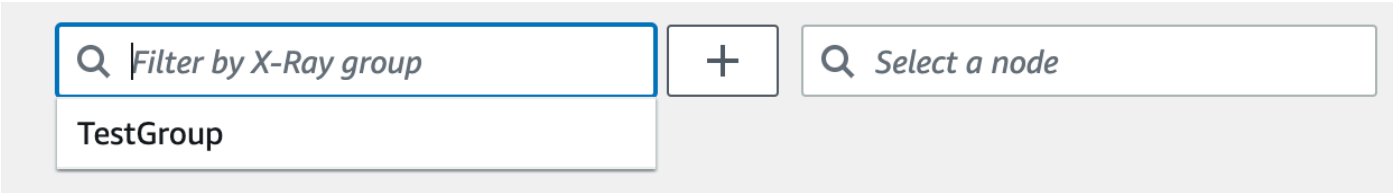
> **Note**
>
> X-Ray 跟踪地图最多可以显示 10,000 个节点。极少数情况下，当服务节点总数超出此上限时，会收到错误消息并且无法在控制台中显示完整的跟踪地图。

按组筛选跟踪地图

通过使用筛选条件表达式，您可以定义某个组中要包含哪些跟踪的标准。然后，使用以下步骤在跟踪地图中显示该特定组。

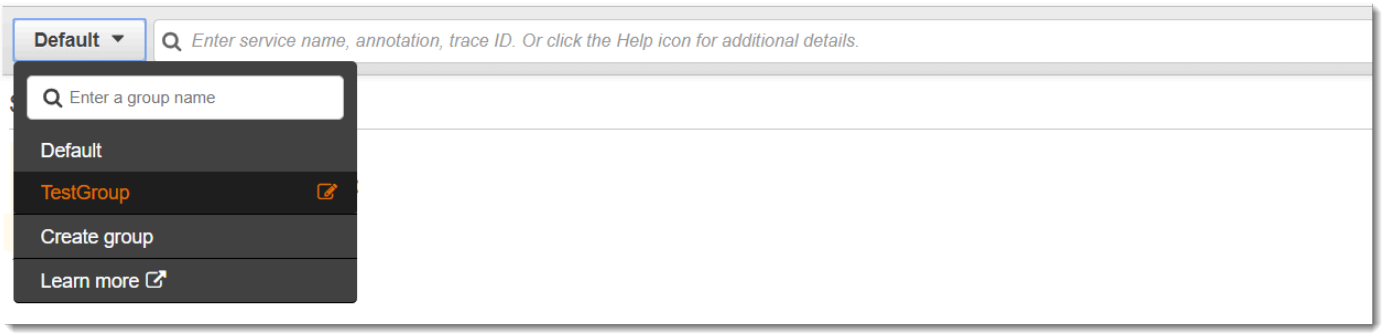
CloudWatch console

从跟踪地图左上角的组筛选器中选择组名称。



X-Ray console

从搜索栏左侧的下拉菜单中选择一个组名称。



现在，将会对服务地图进行筛选以显示与所选组的筛选条件表达式匹配的跟踪。

跟踪地图图例和选项

跟踪地图包含图例和多个选项用于自定义地图显示。

CloudWatch console

选择地图右上角的图例和选项下拉列表。选择节点内显示的内容，其中包括：

- 指标显示所选时间范围内的平均响应时间和每分钟发送的跟踪数量。
- 节点显示每个节点内的服务图标。

从首选项窗格中选择更多地图设置，可通过点击地图右上角的齿轮图标访问。这些设置包括选择使用哪个指标来确定每个节点的大小，以及应在地图上显示哪些 Canary。

X-Ray console

在地图右上角选择地图图例链接，显示服务地图图例。可以在跟踪地图的右下角选择服务地图选项，包括：

- 每人节点内显示的服务图标切换，用于切换是显示服务图标，还是平均呼应时间以及在所选时间范围内每分钟的跟踪数量。
- 节点大小：None 将所有节点设置为相同大小。
- 节点大小：运行状况按受影响的请求数量确定节点大小，其中包括错误、故障或受限制的请求。
- 节点大小：流量按请求总数确定节点大小。

查看跟踪和跟踪详情

使用 X-Ray 控制台中的跟踪页面根据 URL、响应代码或跟踪摘要中的其他数据查找跟踪。从跟踪列表中选择跟踪后，跟踪详情页面会显示与所选跟踪关联的服务节点的地图，以及跟踪分段的时间表。

查看跟踪

CloudWatch console

在 CloudWatch 控制台中查看跟踪

1. 登录 AWS Management Console 并打开 CloudWatch 控制台，网址为https://console.aws.amazon.com/cloudwatch/。
2. 在左侧导航窗格中，依次选择 X-Ray 跟踪和跟踪。您可以按组筛选或输入筛选条件表达式。这将筛选出显示在页面底部跟踪部分中的跟踪。

   或者，也可以使用服务地图导航到某个具体的服务节点，然后查看跟踪。这将打开已应用查询的跟踪页面。
3. 在查询优化部分中优化您的查询。要按常用属性筛选跟踪，请从按条件优化查询旁边的向下箭头中选择一个选项。这些选项包括以下内容：

   - 节点 - 按服务节点筛选跟踪。
   - 资源 ARN - 按与跟踪关联的资源筛选跟踪。这些资源的示例包括亚马逊弹性计算云 (Amazon EC2) 实例、 AWS Lambda 函数或 Amazon DynamoDB 表。
   - 用户 - 按用户 ID 筛选跟踪。
   - 错误根本原因消息 - 按错误根本原因筛选跟踪。
   - URL - 按应用程序使用的 URL 路径筛选跟踪。
   - HTTP 状态码 - 按应用程序返回的 HTTP 状态码筛选跟踪。您可以指定自定义响应代码或从以下代码中进行选择：
     - `200` - 请求成功。
     - `401` - 请求缺少有效的身份验证凭证。
     - `403` - 请求缺少有效权限。
     - `404` - 服务器找不到请求的资源。
     - `500` - 服务器遇到了意外情况并生成了内部错误。

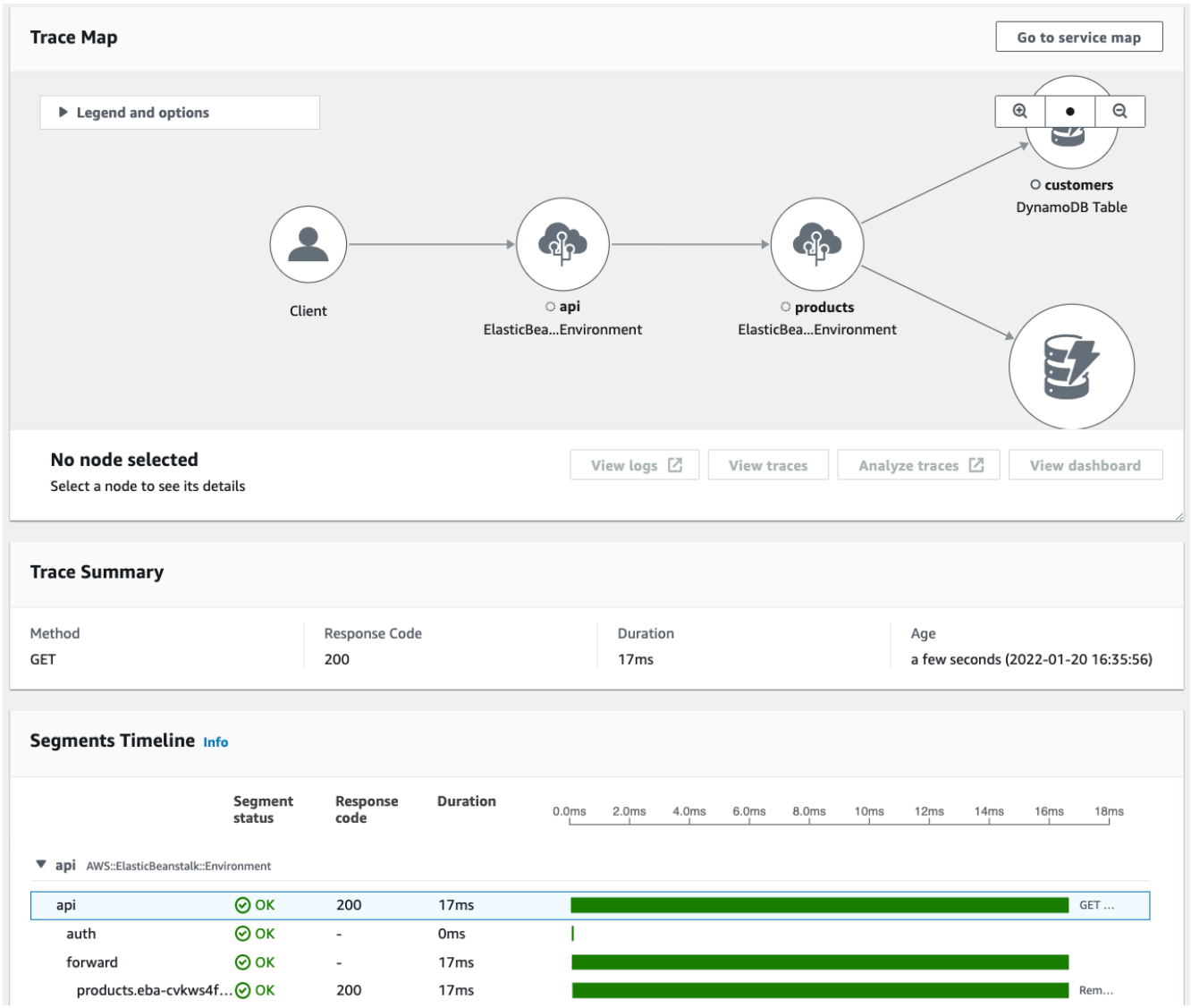
选择一个或多个条目，然后选择添加到查询，将所选条目添加到页面顶部的筛选条件表达式中。

4. 要查找单个跟踪，请直接在查询字段中输入跟踪 ID。可以使用 X-Ray 格式或 World Wide Web Consortium（W3C）格式。例如，使用 AWS Distro for 创建的跟踪采用 OpenTelemetry W3C 格式。

   > **Note**
   >
   > 当您查询采用 W3C 格式跟踪 ID 创建的跟踪时，控制台会显示 X-Ray 格式的匹配跟踪。例如，如果您以 W3C 格式查询 `4efaaf4d1e8720b39541901950019ee5`，控制台会显示等效的 X-Ray：`1-4efaaf4d-1e8720b39541901950019ee5`。

5. 随时选择运行查询，可以在页面底部的跟踪部分中匹配的跟踪列表。
6. 要显示单个跟踪的跟踪详情，请从列表中选择一个跟踪 ID。

下图所示的跟踪地图包含与跟踪关联的服务节点，以及代表构成跟踪的分段所采用路径的节点之间的边缘。跟踪摘要之后是跟踪地图。摘要包含有关 GET 操作示例、其响应代码、跟踪运行持续时间以及请求时限的信息。分段时间线之后是跟踪摘要，该摘要显示跟踪分段和子分段的持续时间。



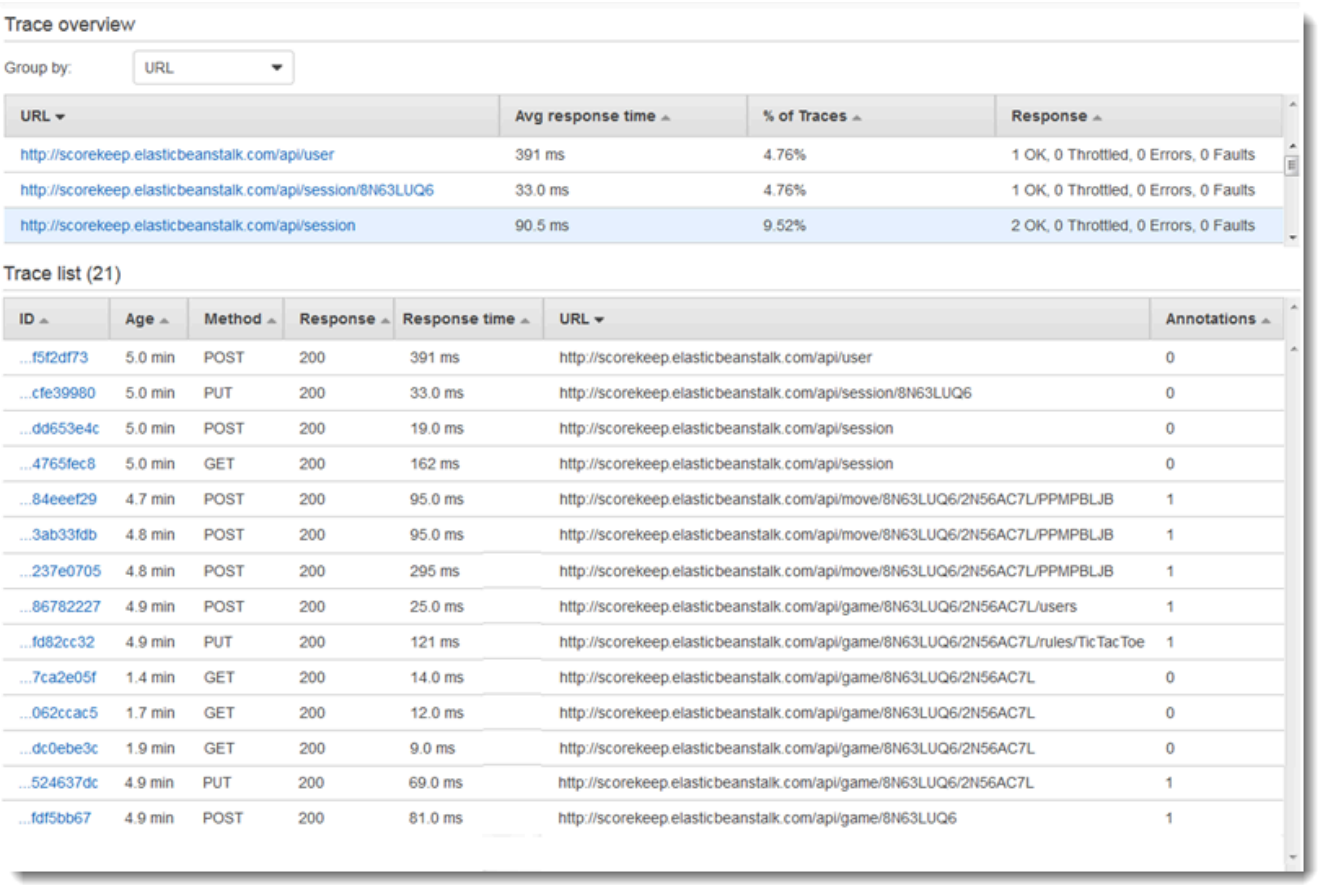
如果您有一个使用 Amazon SQS 和 Lambda 的事件驱动型应用程序，则可以在跟踪地图中看到每个请求的关联跟踪视图。在地图中，来自消息生产者的跟踪链接到来自 AWS Lambda 消费者的跟踪，并显示为虚线边缘。有关事件驱动型应用程序的更多信息，请参阅跟踪事件驱动型应用程序。

跟踪和跟踪详情页面还支持跨账户跟踪，其中会列出跟踪列表中和单个跟踪地图中多个账户内的跟踪。

X-Ray console

如何在 X-Ray 控制台中查看跟踪

1. 在 X-Ray 控制台中打开跟踪页面。跟踪概述面板显示了按常见功能分组的跟踪列表，这些跟踪记录包括错误根本原因、R esourceArn 和。InstanceId
2. 要选择常用特征来查看分组的跟踪集，请展开分组依据旁边的向下箭头。下图显示了按 AWS X-Ray 示例应用程序 URL 分组的跟踪的跟踪概述以及关联跟踪的列表。

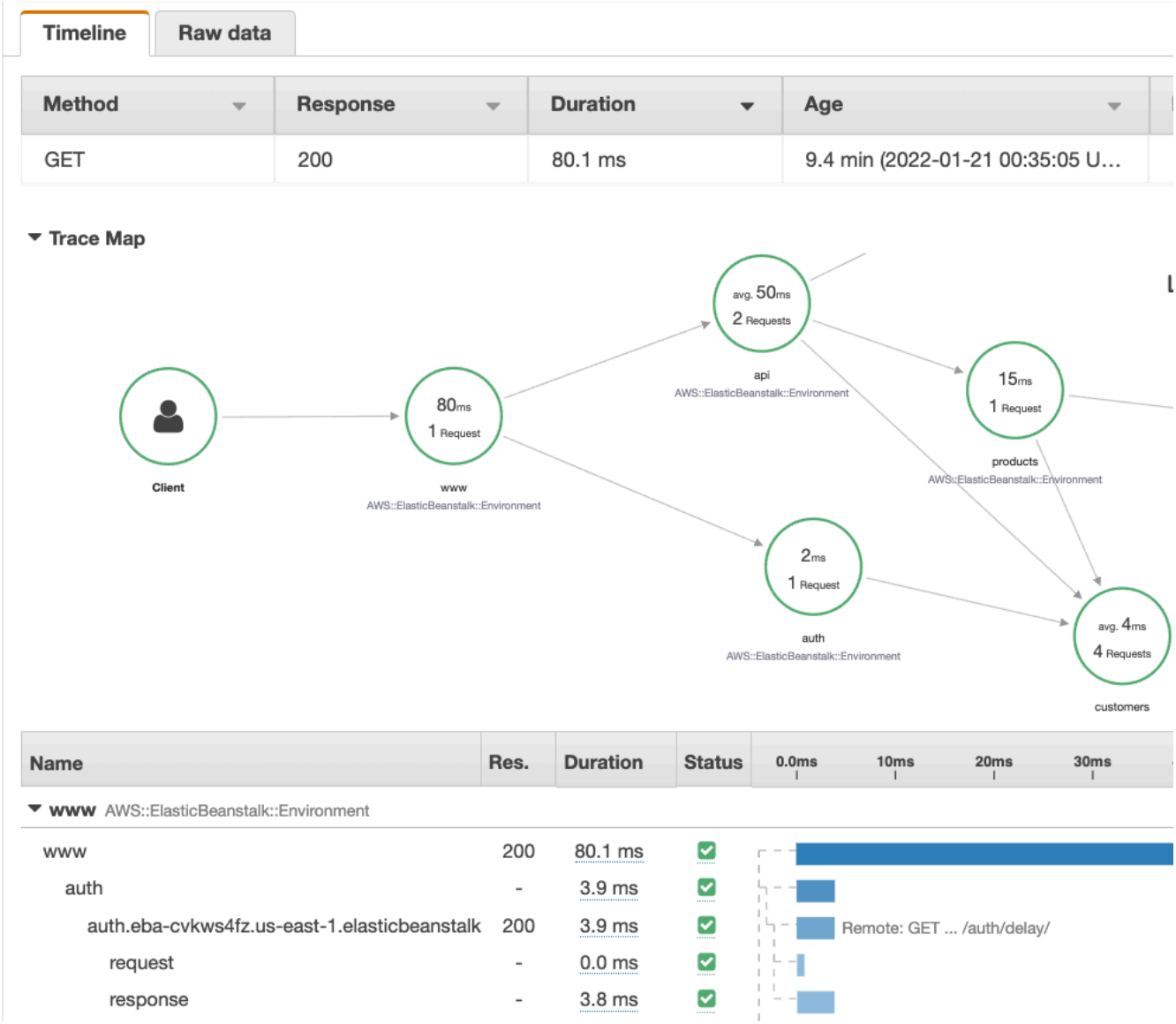


3. 选择跟踪的 ID 以在跟踪列表下查看。您也可以在导航窗格中选择服务地图，来查看特定服务节点的跟踪。然后，您可以查看与该节点关联的跟踪。

   时间线选项卡显示跟踪的请求流，包括以下内容：

   - 跟踪中每个分段的路径地图。
   - 分段到达跟踪地图中的节点花了多长时间。
   - 在跟踪地图中向该节点发出了多少个请求。

下图显示了与向应用程序示例发出的 GET 请求关联的跟踪地图示例。箭头显示每个分段完成请求所采用的路径。服务节点显示在 GET 请求期间发出的请求数。



有关时间线选项卡的更多信息，请参阅下面的深入了解跟踪时间线一节。

原始数据选项卡以 JSON 格式显示有关跟踪以及构成该跟踪的分段和子分段的信息。此类信息可能包含以下内容：

- 时间戳
- 独特 IDs
- 与分段或子分段关联的资源

- 分段或子分段的源或起源
- 有关对您的应用程序发出的请求的其他信息，例如 HTTP 请求的响应

深入了解跟踪时间线

时间线部分在水平条旁边显示分段和子分段的层次结构，该水平条与其完成任务所用的时间相对应。列表中的第一个条目为分段，表示服务为单个请求记录的所有数据。子分段以缩进形式列出，并在分段后面列出。各列包含有关每个分段的信息。

CloudWatch console

在 CloudWatch 控制台中，区段时间轴提供以下信息：

- 第一列：列出所选跟踪中的分段和子分段。
- 分段状态列：列出每个分段和子分段的状态结果。
- 响应代码列：列出对分段或子分段发出的浏览器请求的 HTTP 响应状态代码（如果有）。
- 持续时间列：列出分段或子分段的运行时长。
- 托管位置列：列出运行分段或子分段的命名空间或环境（如果适用）。有关更多信息，请参阅收集的维度和维度组合。
- 最后一列：显示与分段或子分段运行的持续时间相对应的水平条（相对于时间线中的其他分段或子分段）。

要按服务节点对分段和子分段列表进行分组，请打开按节点分组。

X-Ray console

在跟踪详情页面中，选择时间线选项卡，以查看构成跟踪的每个分段和子分段的时间线。

在 X-Ray 控制台中，时间线提供以下信息：

- 名称列：列出跟踪中分段和子分段的名称。
- 响应列：列出针对分段或子分段发出的浏览器请求的 HTTP 响应状态代码（如果有）。
- 持续时间列：列出分段或子分段的运行时长。
- 状态列：列出分段或子分段状态的结果。
- 最后一列：显示与分段或子分段运行的持续时间相对应的水平条（相对于时间线中的其他分段或子分段）。

要查看控制台用来生成时间线的原始跟踪数据，请选择原始数据选项卡。原始数据以 JSON 格式显示有关跟踪以及构成该跟踪的分段和子分段的信息。此类信息可能包含以下内容：

- 时间戳
- 独特 IDs
- 与分段或子分段关联的资源
- 分段或子分段的源或起源
- 有关对您的应用程序发出的请求的其他信息，例如 HTTP 请求的响应。

当你使用装有工具的 S AWS DK 时，HTTP，或 SQL 客户端调用外部资源，X-Ray SDK 会自动记录子分段。也可以使用 X-Ray SDK 记录任何函数或代码块的自定义子分段。自定义子分段在打开时记录的其他子分段将成为自定义子分段的子级。

查看分段详细信息

在跟踪时间线中选择某分段的名称，可以查看其详细信息。

分段详细信息面板显示概述、资源、注释、元数据、异常以及 SQL 选项卡。以下选项卡将适用：

- 概述选项卡显示有关请求和响应的信息。信息包括名称、开始时间、结束时间、持续时间、请求 URL、请求操作、请求响应代码以及任何错误和故障。
- 区段的 “资源” 选项卡显示来自 X-Ray SDK 的信息以及有关运行应用程序的 AWS 资源的信息。使用适用于 X-Ray SDK 的 Amazon EC2 AWS Elastic Beanstalk、或 Amazon ECS 插件来记录特定于服务的资源信息。有关插件的更多信息，请参阅配置 X-Ray SDK for Java中的服务插件部分。
- 其余的选项卡显示为分段记录的注释、元数据和异常。当所检测的请求生成异常时，会自动捕获这些异常。注释和元数据包含您使用 X-Ray SDK 提供的操作记录的附加信息。要将注释或元数据添加到分段，请使用 X-Ray SDK。有关更多信息，请参阅 “使用中的工具化您的应用程序” 下面列出的特定语言链接。 AWS X-Ray SDKs 正在对您的应用程序进行检测 AWS X-Ray

查看子分段详细信息

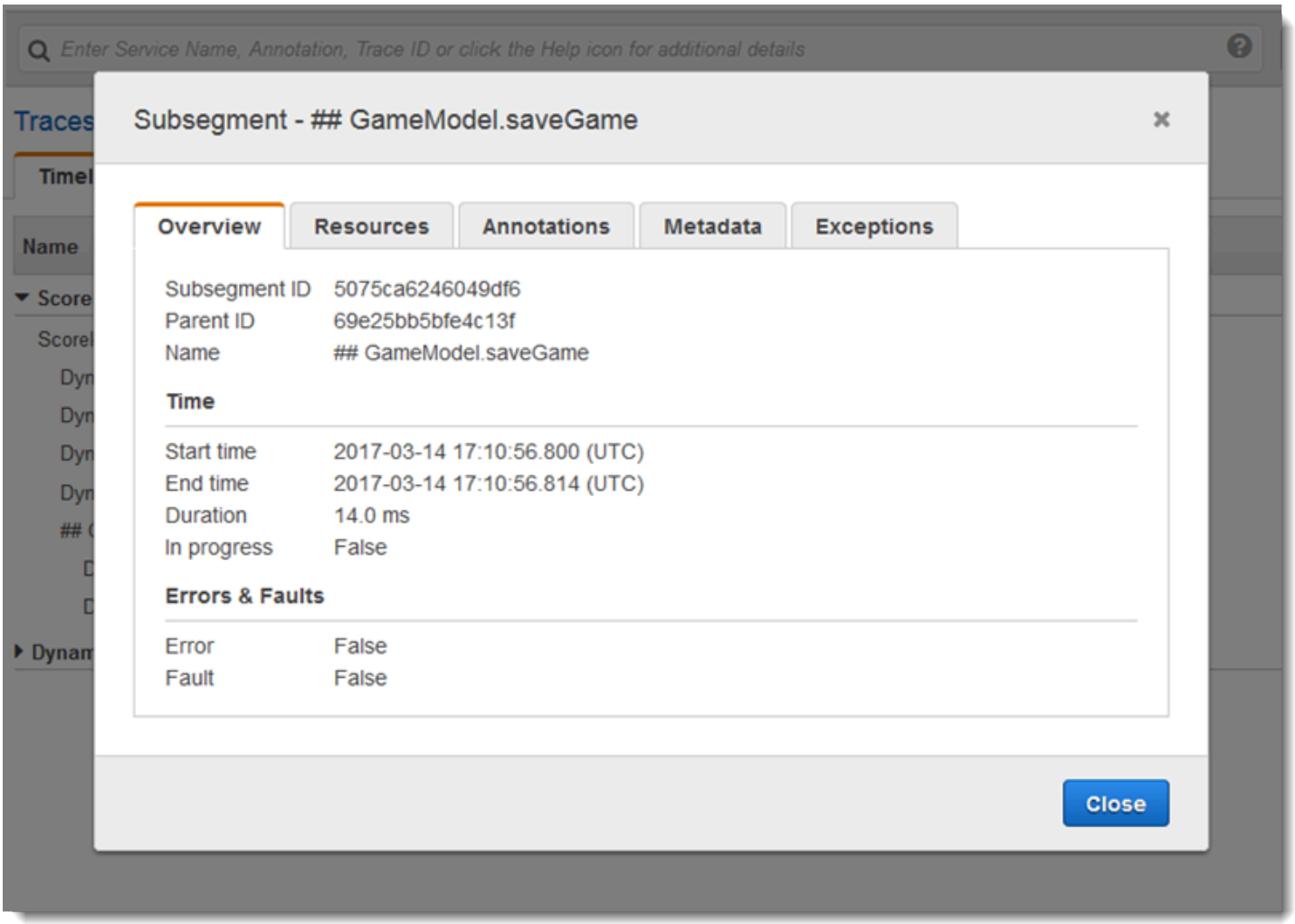
在跟踪时间线中选择某个子分段的名称，可以查看其详细信息。

- 概述选项卡包含有关请求和响应的信息。这包括名称、开始时间、结束时间、持续时间、请求 URL、请求操作、请求响应代码以及任何错误和故障。对于使用已检测客户端生成的子分段，概述选项卡包含从您的应用程序角度来看的请求和响应信息。

- 子分段的资源选项卡显示有关用于运行子分段的 AWS 资源的详细信息。例如，资源选项卡可能包含 AWS Lambda 函数 ARN、有关 DynamoDB 表的信息、调用的任何操作以及请求 ID。
- 其余的选项卡显示子分段上记录的注释、元数据和异常。当所检测的请求生成异常时，会自动捕获这些异常。注释和元数据包含您使用 X-Ray SDK 提供的操作记录的附加信息。使用 X-Ray SDK，将注释或元数据添加到分段。有关更多信息，请参阅 “使用中的工具化您的应用程序” 下面列出的特定语言链接。 AWS X-Ray SDKs 正在对您的应用程序进行检测 AWS X-Ray

对于自定义子分段，概述选项卡显示子分段的名称，您可以设置该名称来指定它所记录的代码或函数区域。有关更多信息，请参阅 “使用中的工具化您的应用程序” 下面列出的特定语言链接。 AWS X-Ray SDKs 使用适用于 Java 的 X-Ray 开发工具包生成自定义子分段

下图显示了自定义子分段的概述选项卡。概述包含子分段 ID、父级 ID、名称、开始和结束时间、持续时间、状态以及错误或故障。



自定义子区段的 “元数据” 选项卡包含以下信息 JSON 关于该子分段使用的资源的格式。

使用筛选条件表达式

使用筛选条件表达式 查看特定请求、服务、两个服务之间的连接（边缘）或满足某个条件的请求的跟踪地图或跟踪。X-Ray 提供筛选表达式语言，根据原始段上请求标头、响应状态和索引字段中的数据筛选请求、服务和边缘。

当您选择某个跟踪时间段以在 X-Ray 控制台中查看时，您获得的结果可能会超出可在控制台中显示的内容。在右上角，控制台显示其扫描的跟踪数量，以及是否有更多跟踪可用。您可以使用筛选条件表达式缩小结果范围，以仅限于您要查找的跟踪。

主题

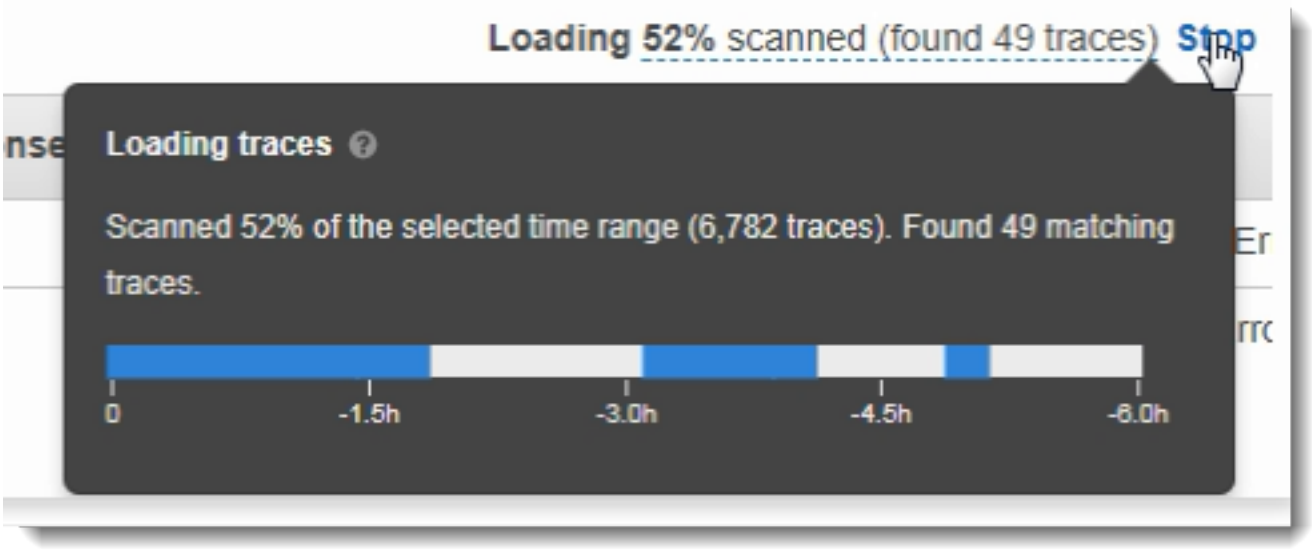
筛选条件表达式详细信息

当您选择跟踪地图中的节点时，控制台会基于该节点的服务名称以及您的选择中提供的错误类型，来构建筛选条件表达式。要查找显示性能问题的跟踪或与特定请求相关的跟踪，可以调整控制台提供的表达式，或创建您自己的表达式。如果您使用 X-Ray SDK 添加注释，您还可以根据是否存在注释键或根据键值进行筛选。

> **Note**
>
> 如果您在跟踪地图中选择相对时间范围并选择一个节点，则控制台会将时间范围转换为绝对开始和结束时间。为了确保节点的跟踪显示在搜索结果中，并避免扫描时间在该节点未处于活动状态的期间内，时间范围只应包含该节点发送跟踪的时间。若要相对于当前时间进行搜索，您可以在跟踪页面中切换回相对时间范围，并重新扫描。

如果结果仍超过控制台可显示的内容，控制台会显示有多少个跟踪匹配，以及扫描的跟踪数。显示的百分比是已扫描选定时间范围的百分比。为确保您会看到在结果中提供所有匹配的跟踪，进一步缩小筛选条件表达式的范围，或选择一个更短的时间范围。

为了先获取最新结果，控制台会从时间范围结尾开始反向扫描。如果有大量的跟踪，但结果很少，控制台会将时间范围分为多个分块并执行并行扫描。进度条显示已扫描的时间范围部分。



将筛选条件表达式与组一起使用

组是由筛选条件表达式定义的跟踪的集合。您可以使用群组来生成其他服务图表并提供 Amazon CloudWatch 指标。

组由其名称或 Amazon 资源名称 (ARN) 标识，并包含筛选条件表达式。此服务将比较传入到表达式的跟踪并相应地存储它们。

您可以使用筛选条件表达式搜索栏左侧的下拉菜单创建和修改组。

> **Note**
>
> 如果服务在限定组时遇到错误，则在处理传入跟踪时不再包含该组，并记录错误指标。

有关组的更多信息，请参阅 配置组。

筛选条件表达式语法

筛选条件表达式可以包含一个关键字、一个一元或二元运算符 和一个值 用于比较。

```
keyword operator value
```

不同的运算符可用于不同类型的关键字。例如，responsetime 是一个数字关键字，可与数字相关运算符进行比较。

Example - 响应时间超过 5 秒的请求

```
responsetime > 5
```

您可以使用 AND 或 OR 运算符将多个表达式组合成一个复合表达式。

Example - 总时长在 5-8 秒之间的请求

```
duration >= 5 AND duration <= 8
```

简单的关键字和运算符只在跟踪级别查找问题。如果下游发生了错误，但被您的应用程序处理了而未返回给用户，则搜索 error 将找不到它。

要查找下游问题的跟踪，可以使用复杂关键字 service() 和 edge()。这些关键字允许您将筛选条件表达式应用于所有下游节点、单个下游节点或两个节点之间的边缘。要想获得更细的粒度，您可以使用 id() 函数按类型筛选服务和边缘。

布尔值关键字

布尔关键字值可为 true 或 false。使用这些关键字查找导致错误的跟踪。

布尔值关键字

- ok - 响应状态代码为 2XX，成功。
- error - 响应状态代码为 4XX，客户端错误。
- throttle - 响应状态代码为“429 请求过多”。
- fault - 响应状态代码为 5XX，服务器错误。
- partial - 请求包含未完成的分段。
- inferred - 请求具有推断分段。
- first - 元素是枚举列表中的第一个元素。
- last - 元素是枚举列表中的最后一个元素。

- `remote` - 根本原因实体是远程的。
- `root` - 服务是跟踪的入口点或根分段。

布尔运算符查找指定键为 `true` 或 `false` 的分段。

布尔运算符

- none - 如果关键字为 true，则表达式为 true。
- ！- 如果关键字为 false，则表达式为 true。
- =、!= - 将关键字的值与字符串 `true` 或 `false` 进行比较。这些运算符与其他运算符的行为相同，但更加明确。

Example - 响应状态为 2XX OK

```
ok
```

Example - 响应状态不为 2XX OK

```
!ok
```

Example - 响应状态不为 2XX OK

```
ok = false
```

Example - 上次枚举的错误跟踪具有错误名称“deserialize”

```
rootcause.fault.entity { last and name = "deserialize" }
```

Example - 包含远程分段的请求，其覆盖率大于 0.7 且服务名称为“traces”

```
rootcause.responsetime.entity { remote and coverage > 0.7 and name = "traces" }
```

Example - 具有推断分段（其中，服务类型为“AWS:DynamoDB”）的请求

```
rootcause.fault.service { inferred and name = traces and type = "AWS::DynamoDB" }
```

Example - 将名称为“data-plane”的分段用作根的请求

```
service("data-plane") {root = true and fault = true}
```

数字关键字

使用数字关键字可以搜索具有特定响应时间、持续时间或响应状态的请求。

数字关键字

- `responsetime` - 服务器发送响应所用的时间。
- `duration` - 包括所有下游调用的请求总时长。
- `http.status` - 响应（状态代码）。
- `index` - 元素在枚举列表中的位置。
- `coverage` - 实体响应时间占根分段响应时间的小数百分比。仅适用于响应时间根本原因实体。

数字运算符

数字关键字使用标准相等运算符和比较运算符。

- =、!= - 关键字等于或不等于某个数值。
- <、<=、>、>= - 关键字小于或大于某个数值。

Example - 响应状态不为 200 OK

```
http.status != 200
```

Example - 总时长在 5-8 秒之间的请求

```
duration >= 5 AND duration <= 8
```

Example - 在 3 秒内成功完成的请求，包括所有下游调用

```
ok !partial duration <3
```

Example - 索引大于 5 的枚举列表实体

```
rootcause.fault.service { index > 5 }
```

Example - 其最后一个实体覆盖率大于 0.8 的请求

```
rootcause.responsetime.entity { last and coverage > 0.8 }
```

字符串关键字

使用字符串关键字查找请求标头中包含特定文本或特定用户的跟踪 IDs。

字符串关键字

- `http.url` - 请求 URL。
- `http.method` - 请求方法。
- `http.useragent` - 请求的用户代理字符串。
- `http.clientip` - 请求者 IP 地址。
- `user` - 跟踪中任意分段的用户字段的值。
- `name` - 服务或异常的名称。
- `type` - 服务类型。
- `message` - 异常消息。
- `availabilityzone` - 跟踪中任意分段上可用区字段的值。
- `instance.id` - 跟踪中任意分段上的实例 ID 字段的值。
- `resource.arn` - 跟踪中任何分段上的资源 ARN 字段的值。

字符串运算符查找等于或包含特定文本的值。必须始终在引号中指定值。

字符串运算符

- `=`、`!=` - 关键字等于或不等于某个数值。
- `CONTAINS` - 关键字包含特定字符串。
- `BEGINSWITH`、`ENDSWITH` - 关键字以特定字符串开头或结尾。

Example - http.url 筛选器

```
http.url CONTAINS "/api/game/"
```

要测试跟踪中是否存在某个字段而不考虑其值，可检查它是否包含空字符串。

Example - 用户筛选器

与用户一起查找所有痕迹 IDs。

```
user CONTAINS ""
```

Example - 选择跟踪，跟踪所具有的故障根本原因包含名为“Auth”的服务

```
rootcause.fault.service { name = "Auth" }
```

Example - 选择跟踪，跟踪所具有的响应时间根本原因的最后一个服务的类型为 DynamoDB

```
rootcause.responsetime.service { last and type = "AWS::DynamoDB" }
```

Example - 选择跟踪，跟踪所具有的故障根本原因的最后一个异常具有消息“拒绝 account_id 访问：1234567890”

```
rootcause.fault.exception { last and message = "Access Denied for account_id:
 1234567890"
```

复杂关键字

使用复杂关键字可根据服务名称、边缘节点名称或注释值查找请求。对于服务和边缘节点，您可以指定应用于服务或边缘节点的附加筛选条件表达式。对于注释，您可以使用布尔值、数字或字符串运算符筛选具有特定键的注释的值。

复杂关键字

- `annotation[`*`key`*`]`-带字段的注释的值*`key`*。注释的值可以是布尔值、数字或字符串，因此您可以使用任意这些类型的比较运算符。此关键字可以与 `service` 或 `edge` 关键字组合使用。包含点（句点）的注释键必须用方括号（[]）括住。
- `edge(`*`source`*`, `*`destination`*`) {`*`filter`*`}`—服务*`source`*与之间的连接*`destination`*. 可选的大括号中可以包含应用于此连接上的分段的筛选条件表达式。
- `group.`*`name`* / `group.`*`arn`* — 组的筛选条件表达式的值，被组名称或组 ARN 所引用。
- `json` - JSON 根本原因对象。有关以编程方式创建 JSON 实体的步骤，请参阅从 AWS X-Ray 获取数据。
- `service(`*`name`*`) {`*`filter`*`}`— 带有名称的服务*`name`*。可选的大括号中可以包含应用于服务所创建的分段的筛选条件表达式。

使用服务关键字查找命中跟踪地图上特定节点的请求的跟踪。

复杂关键字运算符可查找其中的指定键已经设置或未设置的分段。

复杂关键字运算符

- none - 如果关键字已经设置，则表达式为 true。如果关键字为布尔类型，则其计算结果将为布尔值。
- ！- 如果关键字未设置，则表达式为 true。如果关键字为布尔类型，则其计算结果将为布尔值。
- =、!= — 比较关键字的值。
- edge(*source*, *destination*) {*filter*}—服务*source*与之间的连接*destination*. 可选的大括号中可以包含应用于此连接上的分段的筛选条件表达式。
- annotation[*key*]-带字段的注释的值*key*。注释的值可以是布尔值、数字或字符串，因此您可以使用任意这些类型的比较运算符。此关键字可以与 service 或 edge 关键字组合使用。
- json - JSON 根本原因对象。有关以编程方式创建 JSON 实体的步骤，请参阅从 AWS X-Ray 获取数据。

使用服务关键字查找命中跟踪地图上特定节点的请求的跟踪。

Example - 服务筛选器

包括对 api.example.com 调用的请求出错 (500 系列错误)。

```
service("api.example.com") { fault }
```

您可以排除服务名称，而将筛选条件表达式应用于服务地图上的所有节点。

Example - 服务筛选器

在跟踪地图上的任意位置导致故障的请求。

```
service() { fault }
```

边缘关键字将筛选条件表达式应用于两个节点之间的连接。

Example - 边缘筛选器

服务 api.example.com 对 backend.example.com 进行调用的请求因出现错误而失败。

```
edge("api.example.com", "backend.example.com") { error }
```

您也可以将 ！ 运算符与服务和边缘关键字结合使用来从另一个筛选条件表达式的结果中排除某个服务或边缘。

Example - 服务和请求筛选器

请求的 URL 以 `http://api.example.com/` 开头且包含 `/v2/`，但并未到达名为 `api.example.com` 的服务。

```
http.url BEGINSWITH "http://api.example.com/" AND http.url CONTAINS "/v2/" AND !
service("api.example.com")
```

Example — 服务和响应时间筛选器

查找已设置 `http url` 且呼应时间大于 2 秒的跟踪。

```
http.url AND responseTime > 2
```

对于注释，您可以调用设置了 `annotation[`*`key`*`]` 的所有跟踪，或使用对应于值的类型的比较运算符。

Example - 带字符串值的注释

请求的注释名为 `gameid`，字符串值为 `"817DL6VO"`。

```
annotation[gameid] = "817DL6VO"
```

Example — 注释已设置

带有名称设置为 `age` 的注释的请求。

```
annotation[age]
```

Example — 注释未设置

不带有名称设置为 `age` 的注释的请求。

```
!annotation[age]
```

Example - 带数字值的注释

请求的注释期限数值大于 29。

```
annotation[age] > 29
```

Example — 注释与服务或边缘相结合

```
service { annotation[request.id] = "917DL6VO" }
```

```
edge { source.annotation[request.id] = "916DL6VO" }
```

```
edge { destination.annotation[request.id] = "918DL6VO" }
```

Example — 带有用户的组

其的跟踪满足 high_response_time 组筛选条件（例如，responseTime > 3），且用户名为“Alice”的请求。

```
group.name = "high_response_time" AND user = "alice"
```

Example - 具有根本原因实体的 JSON

具有匹配的根本原因实体的请求。

```
rootcause.json = #[{ "Services": [ { "Name": "GetWeatherData", "EntityPath": [{ "Name":
 "GetWeatherData" }, { "Name": "get_temperature" } ] }, { "Name": "GetTemperature",
 "EntityPath": [ { "Name": "GetTemperature" } ] } ] }]
```

id 函数

当您为 service 或 edge 关键字提供服务名称时，您将得到具有该名称的所有节点的结果。要进行更精确的筛选，可以使用 id 函数在名称之外再指定一个服务类型，以区分同名节点。

在监控账户中查看多个账户中的跟踪时，使用 account.id 函数为服务指定一个具体账户。

```
id(name: "service-name", type:"service::type", account.id:"account-ID")
```

您可以在服务和边缘节点筛选条件中使用 id 函数来代替服务名称。

```
service(id(name: "service-name", type:"service::type")) { filter }
```

```
edge(id(name: "service-one", type:"service::type"), id(name: "service-two",
 type:"service::type")) { filter }
```

例如，AWS Lambda 函数会在跟踪映射中生成两个节点；一个用于函数调用，另一个用于 Lambda 服务。两个节点的名称相同，但类型不同。标准服务筛选器将查找这两个节点的跟踪。

Example - 服务筛选器

在任何名为 random-name的服务上包含错误的请求。

```
service("random-name") { error }
```

使用 id 函数将搜索范围缩小到函数本身的错误，排除服务的错误。

Example - 使用 id 函数的服务筛选器

名为 random-name、类型为 AWS::Lambda::Function 的服务中有错误的请求。

```
service(id(name: "random-name", type: "AWS::Lambda::Function")) { error }
```

要按类型搜索节点，您还可以完全排除名称。

Example — 具有 id 函数和服务类型的服务筛选器

类型为 AWS::Lambda::Function 的服务中有错误的请求。

```
service(id(type: "AWS::Lambda::Function")) { error }
```

要搜索特定节点 AWS 账户，请指定账户 ID。

Example - 具有 id 函数和账户 ID 的服务筛选器

包含某个特定账户 ID AWS::Lambda::Function 中某项服务的请求。

```
service(id(account.id: "account-id"))
```

跨账户跟踪

AWS X-Ray 支持跨账户可观察性，使您能够监控跨多个账户的应用程序并对其进行故障排除。 AWS 区域您可以如同在一个账户中进行操作那样，无缝搜索、可视化和分析任何关联账户中的指标、日志和跟踪。这样可提供在多个账户之间移动的请求的完整视图。您可以在 X-Ray 跟踪地图中查看跨账户跟踪，也可以在CloudWatch控制台中查看跟踪页面。

共享的可观测性数据可以包括以下任意类型的遥测数据：

- Amazon 中的指标 CloudWatch
- Amazon CloudWatch 日志中的日志组
- 有痕迹进来 AWS X-Ray
- Amazon CloudWatch 应用程序见解中的应用程序

配置跨账户可观测性

要开启跨账户可观察性，请设置一个或多个 AWS 监控账户，并将它们与多个来源账户关联。监控账户是一个 AWS 账户 可以查看源账户生成的可观测性数据并与之交互的中心。源账户是指为 AWS 账户 其所包含的资源生成可观测性数据的个人。

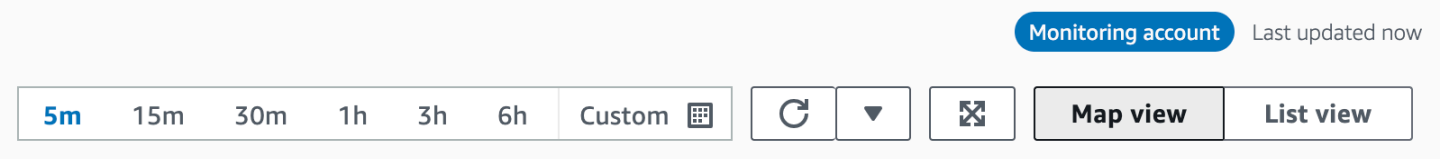
源账户与监控账户共享其可观测性数据。最多可将每个源账户中的跟踪复制到 5 个监控账户。将源账户中的跟踪副本到第一个监控账户免费。发送到更多监控账户的副本根据标准定价，向每个源账户收费。有关更多信息，请参阅AWS X-Ray 定价和 Amazon CloudWatch 定价。

要在监控账户和源账户之间创建链接，请使用 CloudWatch 控制台或和 API 中的全新 Observability Access Manager 命令。 AWS CLI 有关更多信息，请参阅 CloudWatch 跨账户可观测性。

> **Note**
>
> X-Ray 追踪按接收 AWS 账户 地点计费。如果采样请求跨越多个服务 AWS 账户，则每个账户都会记录一条单独的跟踪，并且所有跟踪共享相同的跟踪 ID。要了解有关跨账户可观察性定价的更多信息，请参阅定价和 A ma CloudWatch zon AWS X-Ray 定价。

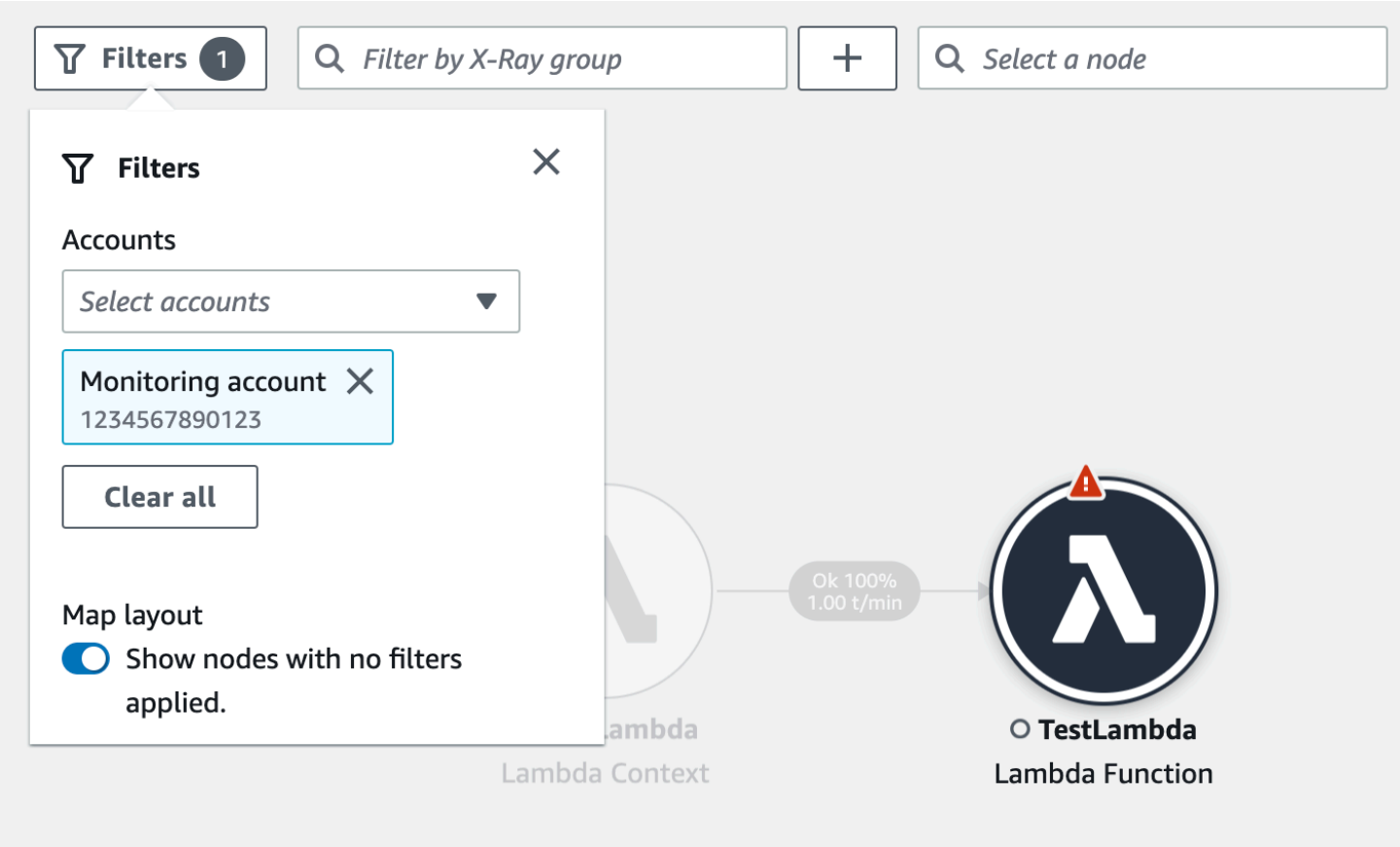
查看跨账户跟踪

跨账户跟踪显示在监控账户中。每个源账户仅显示该特定账户的本地跟踪。以下各节假设您已登录监控账户并已打开 Amazon CloudWatch 控制台。在跟踪地图和跟踪页面上，监控账户徽章都显示在右上角。

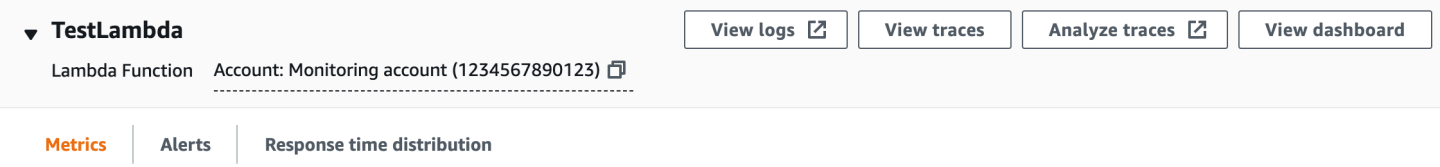


跟踪地图

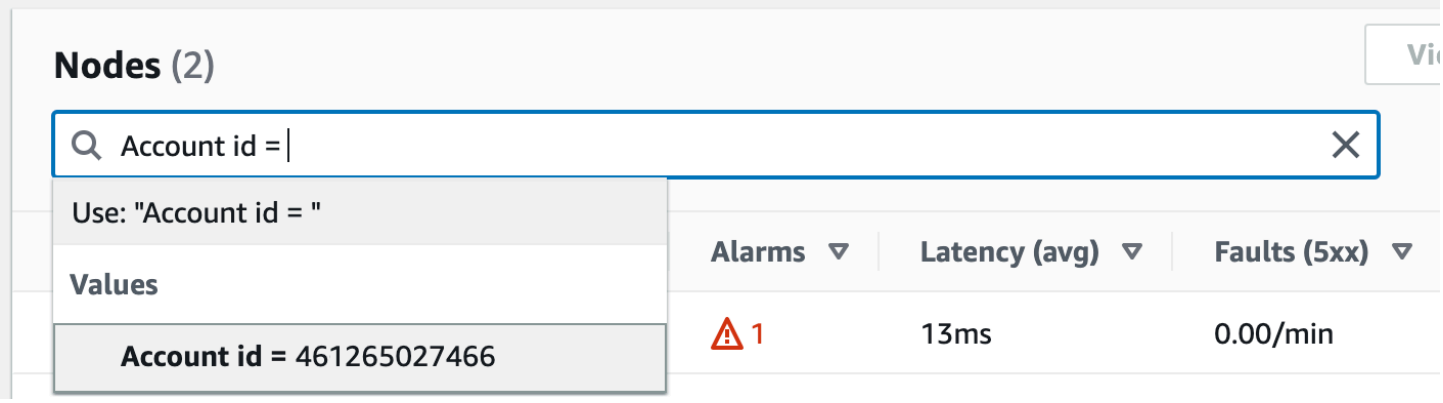
在 CloudWatch 控制台中，从左侧导航窗格的 X-Ray 轨迹下选择 Trace Map。默认情况下，跟踪地图显示将跟踪发送到监控账户的所有源账户的节点，以及监控账户本身的节点。在跟踪地图上，选择左上角的筛选条件，使用账户下拉列表筛选跟踪地图。应用账户筛选条件后，与当前筛选条件不匹配的账户的服务节点将显示为灰色。



选择服务节点时，节点详细信息窗格将包含该服务的账户 ID 和标签。



在跟踪地图的右上角，选择列表视图以查看服务节点列表。服务节点列表包括来自监控账户的服务以及所有已配置的源帐户。从节点筛选条件中进行选择，按 Account label 或 Account id 筛选节点列表。

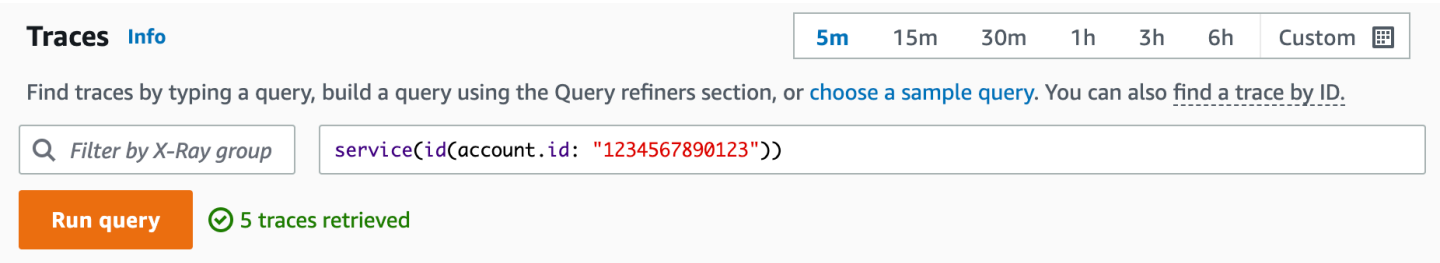


跟踪

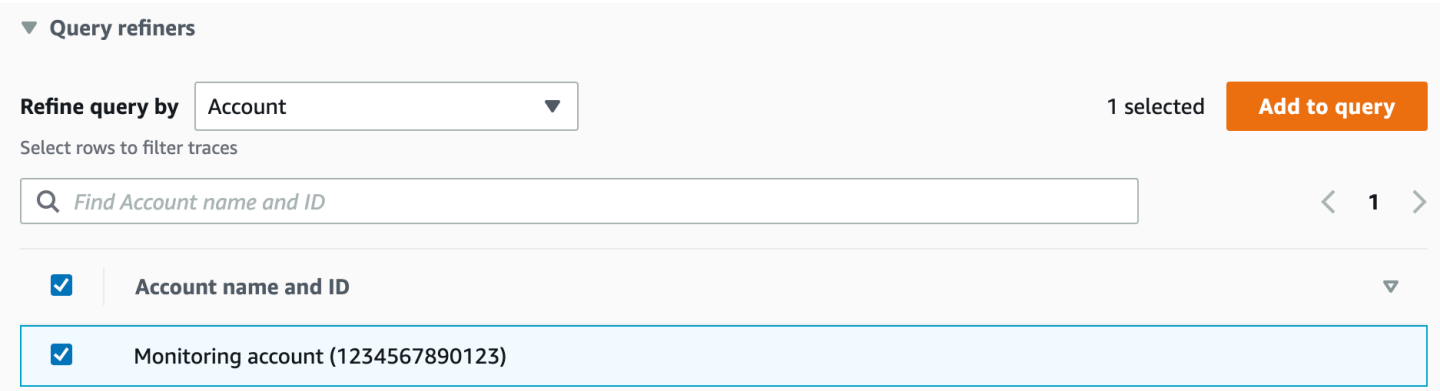
通过从监控账户打开 CloudWatch 控制台，然后在左侧导航窗格的 X-Ray traces 下选择跟踪，即可查看跨多个账户的跟踪详情。也可以通过在 X-Ray 跟踪地图中选择一个节点，然后从节点详细信息窗格中选择查看跟踪来打开此页面。

跟踪页面支持按账户 ID 进行查询。首先，请输入包含一个或多个账户的查询 IDs。以下示例查询通过账户 ID X 或 Y 的跟踪：

```
service(id(account.id:"X")) OR service(id(account.id:"Y"))
```



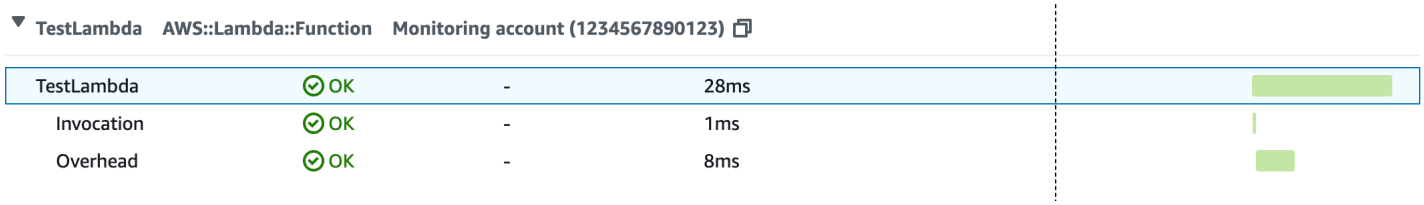
按账户优化查询。从列表中选择一个或多个账户，然后选择添加到查询。



跟踪详情

从跟踪页面底部的跟踪列表中选择相应跟踪，可查看该跟踪的详细信息。会显示跟踪详情，其中包括一张跟踪详情地图，其中包含相应跟踪通过的所有账户中的服务节点。选择某个具体的服务节点查看其相应的账户。

分段时间线一节按照时间线显示每个分段的账户详细信息。



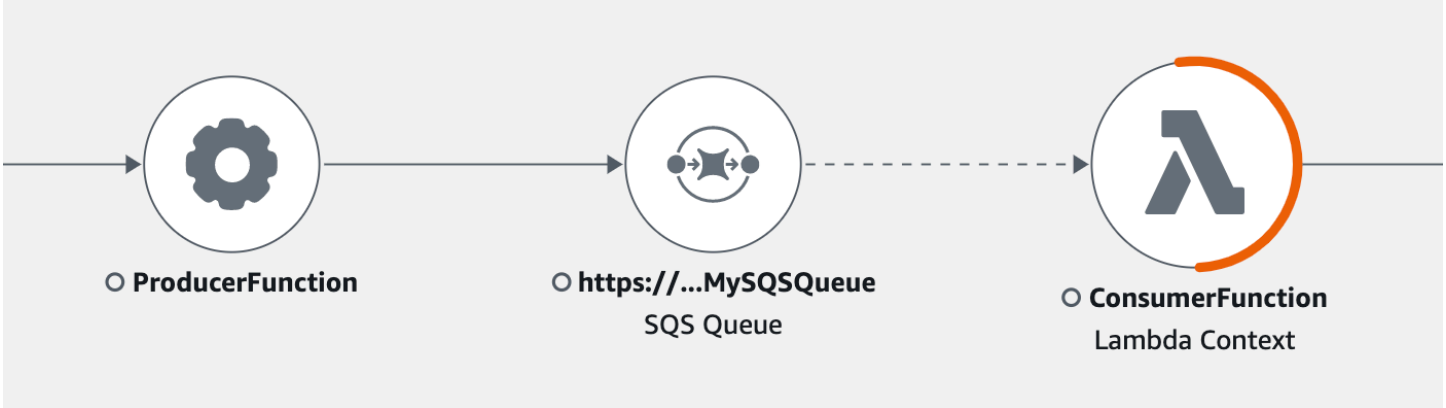
跟踪事件驱动型应用程序

AWS X-Ray 支持使用 Amazon SQS 和跟踪事件驱动的应用程序。 AWS Lambda使用 CloudWatch 控制台查看每个请求在 Amazon SQS 中排队并由一个或多个 Lambda 函数处理的连接视图。来自上游消息生成者的跟踪会自动链接到来自下游 Lambda 使用者节点的跟踪，从而创建 end-to-end应用程序视图。

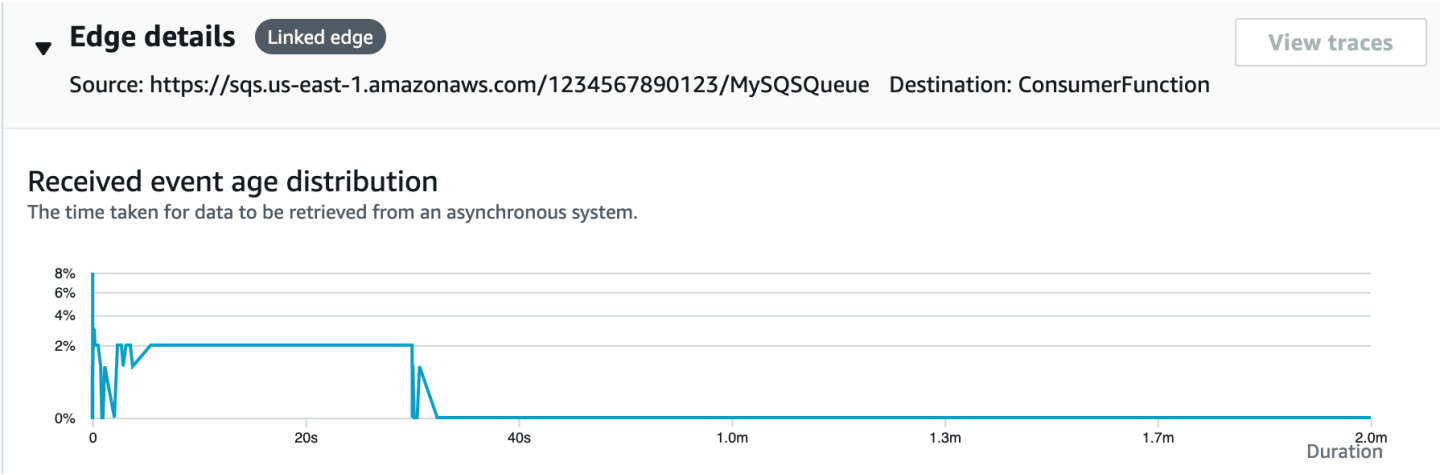
> **Note**
>
> 每个跟踪分段最多可以链接到 20 个跟踪，每个跟踪最多可包含 100 个链接。某些情况下，链接更多跟踪可能会导致超出最大的跟踪文档大小，可能会造成跟踪不完整。例如，当启用了跟踪的 Lambda 函数在一次调用中将许多 SQS 消息发送到一个队列会发生这种情况。如果您遇到此问题，可以使用使用 X-Ray 的缓解措施 SDKs。有关更多信息，请参阅适用于 Java、Node.js、Python、Go 或 .NET 的 X-Ray SDK。

在跟踪地图中查看链接的跟踪

使用CloudWatch控制台中的 Trace Map 页面查看跟踪地图，其中包含来自消息生成者的跟踪，这些跟踪链接到来自 Lambda 使用者的跟踪。这些链接以虚线边缘显示，连接到 Amazon SQS 节点和下游 Lambda 使用器节点。



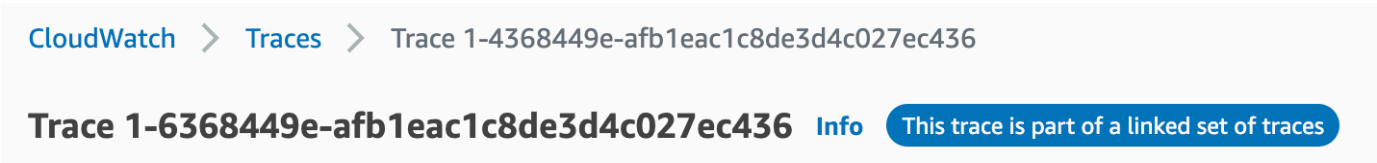
选择虚线边缘以显示收到的事件期限直方图，图中显示了使用器收到时事件年限的分布情况。每次收到事件时都会计算期限。



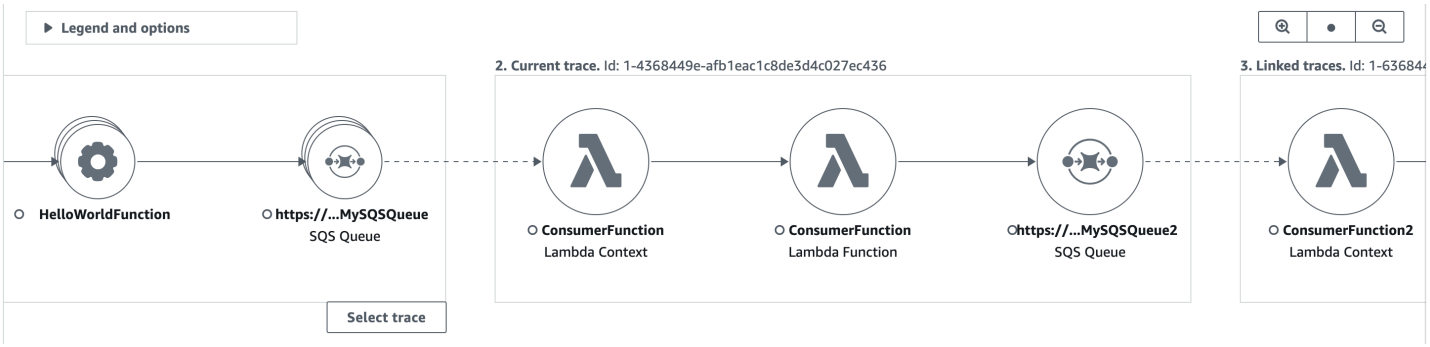
查看链接的跟踪详情

查看消息创建者、Amazon SQS 队列或 Lambda 使用器发送的跟踪详情：

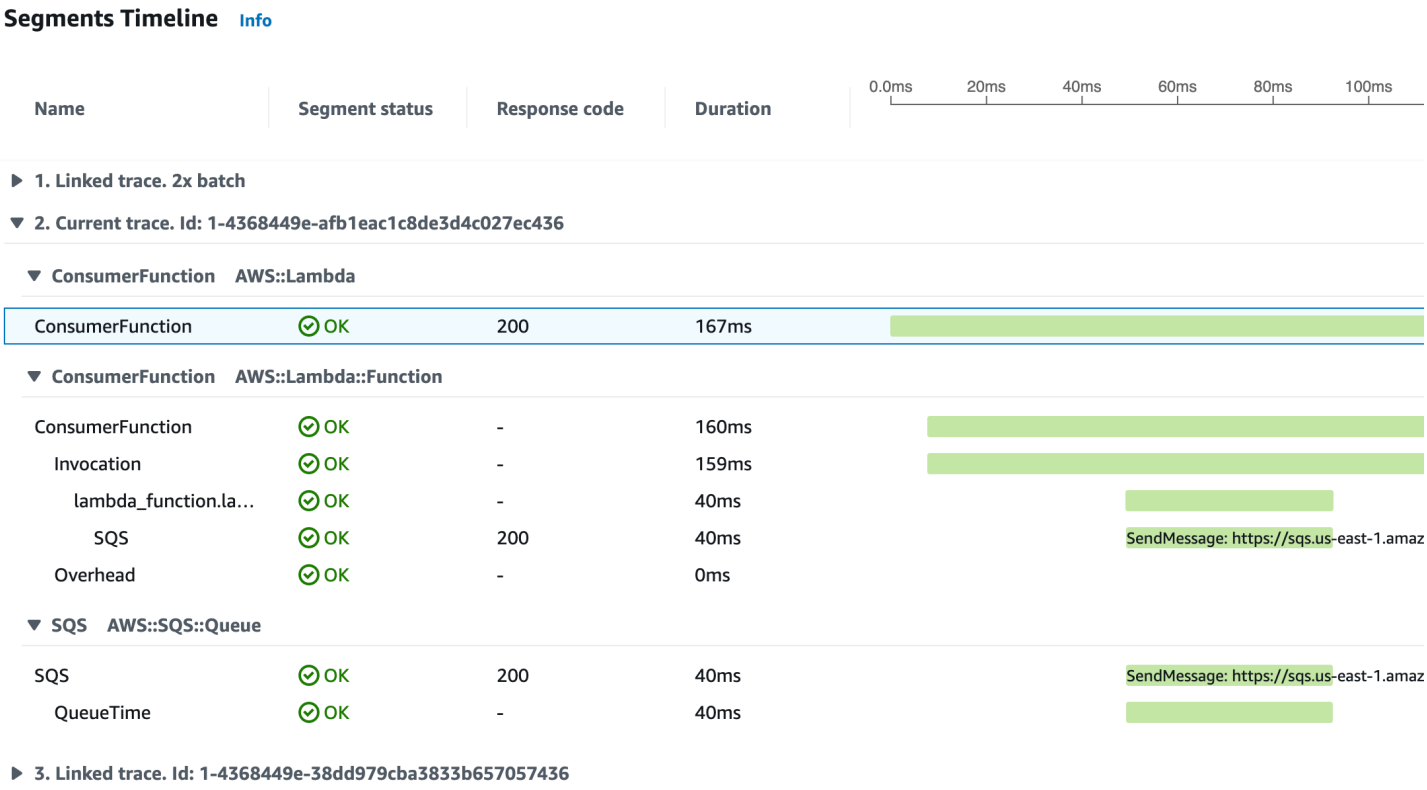
1. 使用跟踪地图选择消息创建者、Amazon SQS 或 Lambda 使用者节点。
2. 从节点详情中选择查看跟踪以显示跟踪列表。您也可以直接导航到 CloudWatch 控制台中的 Traces 页面。
3. 从列表中选择特定跟踪以打开跟踪详情页面。跟踪详情页面显示所选跟踪是链接的跟踪集合的一部分时的消息。



跟踪详情地图显示当前跟踪以及上下游链接的跟踪，其中每个跟踪都包含在指示每个跟踪边界的框中。如果当前选择的跟踪链接到多个上游或下游跟踪，则上游或下游链接的跟踪的节点会堆叠在一起，并会显示选择跟踪按钮。



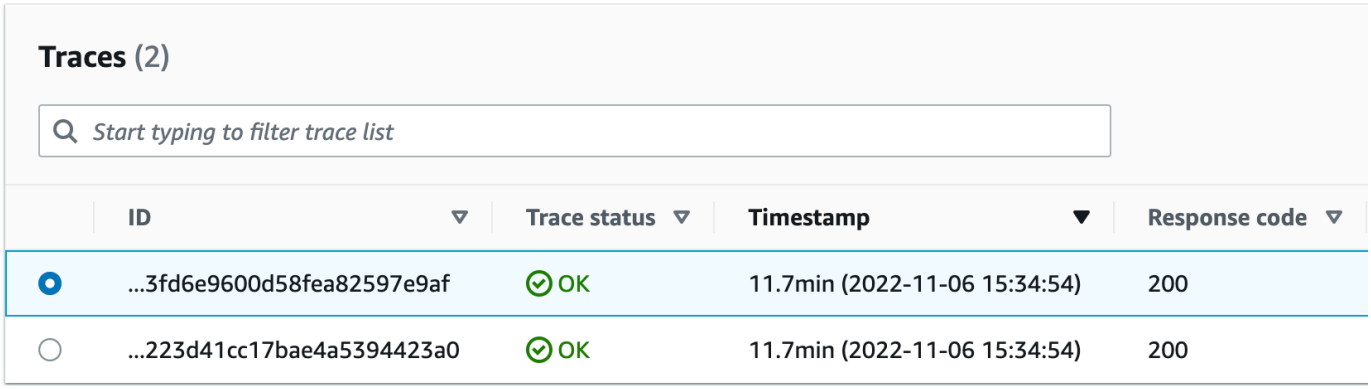
在跟踪详情地图下方显示跟踪分段的时间表，其中包含上下游链接的跟踪。如果有多个上游或下游链接的中，则不会显示它们的分段详情。若要查看链接的跟踪集合中某一个跟踪的分段详情，选择单一跟踪，如下所述。



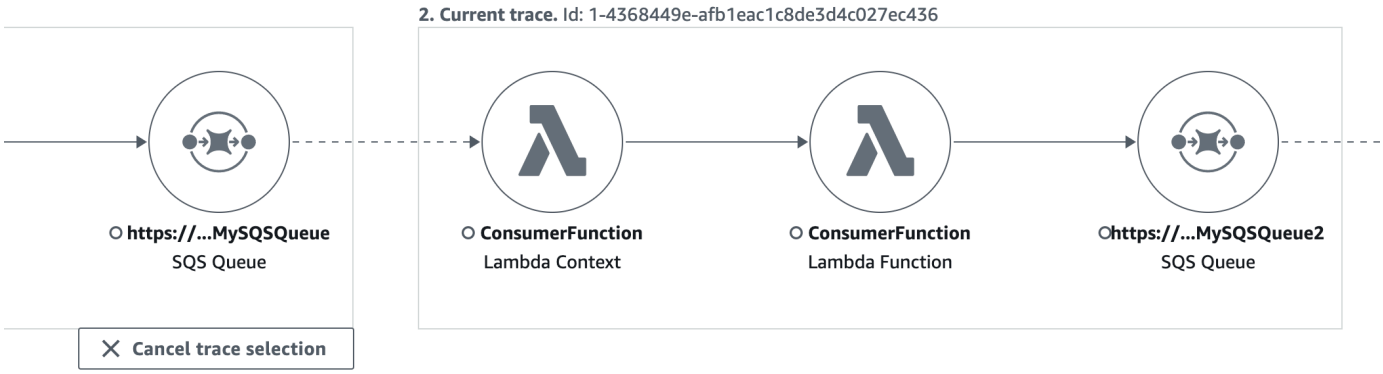
选择链接的跟踪集合中的某一个跟踪

将链接的跟踪集合筛选到只有一个跟踪，以时间表的形式查看分段详情。

1. 在跟踪详情地图中链接的跟踪下方，选择选择跟踪。将会显示跟踪列表。



2. 选中跟踪旁边的单选按钮，在跟踪详情地图里查看它。
3. 选择取消跟踪选择以查看链接的跟踪的整个集合。



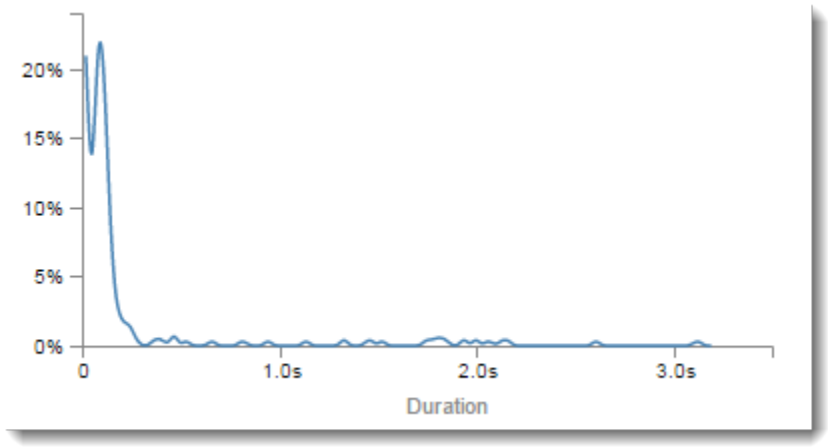
使用延迟直方图

当您在 AWS X-Ray 跟踪地图上选择节点或边缘时，X-Ray 控制台会显示延迟分布直方图。

延迟

延迟是指请求从开始到完成所用的时间。直方图显示延迟分布。它的 x 轴显示持续时间，y 轴显示与每个持续时间匹配的请求百分比。

该直方图显示服务在不到 300 ms 的时间内完成大多数请求。一小部分请求用时多达 2 秒，而一些异常项用了更多时间。



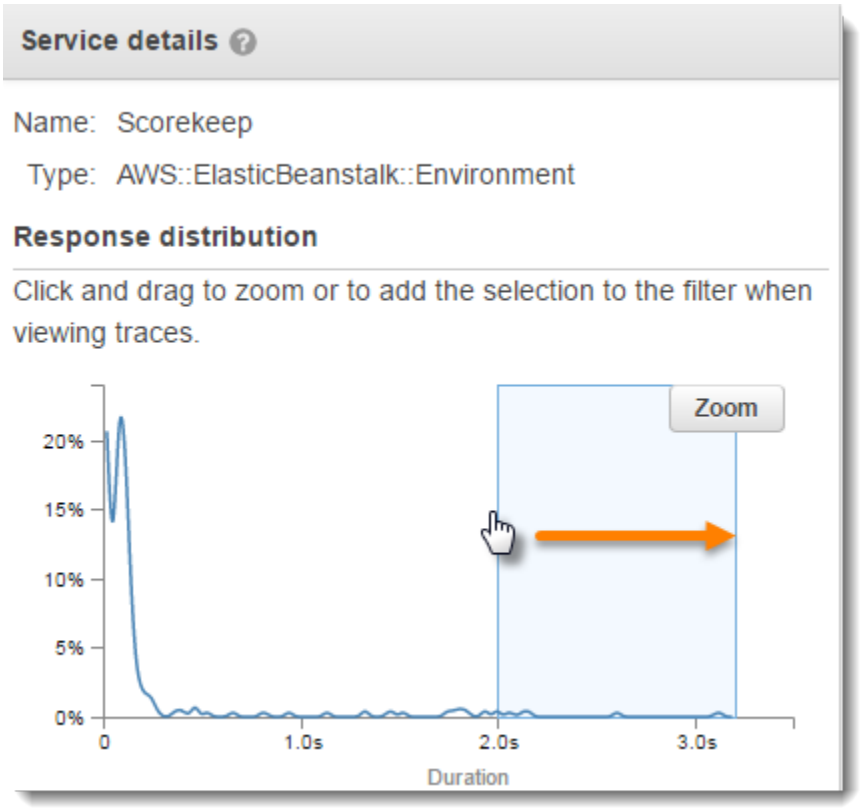
解释服务详细信息

服务直方图和边缘直方图提供了从服务或请求者角度看的延迟的可视化表示形式。

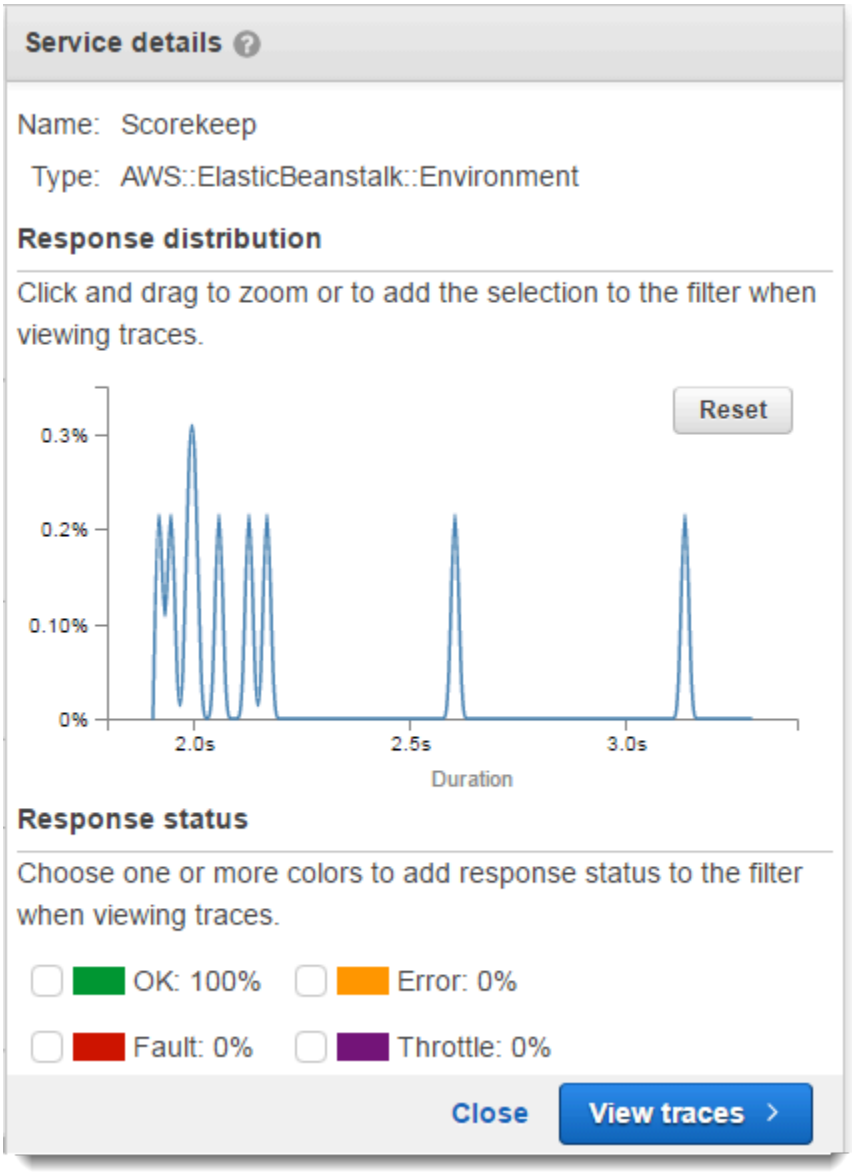
- 通过单击圆圈选择服务节点。X-Ray 显示服务所完成的请求的直方图。延迟是服务记录的延迟，不包括服务和请求者之间的任何网络延迟。
- 通过单击两个服务之间边缘的线条或箭头尖端来选择边缘。X-Ray 显示来自请求者的、由下游服务所完成的请求的直方图。延迟是服务记录的延迟，且包括两个服务之间的网络连接延迟。

要解释服务详细信息面板直方图，您可以查找偏离直方图中大多数值最大的值。可以将这些异常值 视为直方图中的高峰或峰值，并且您可以查看特定区域的跟踪以调查发生了什么情况。

要查看按延迟筛选的跟踪，请在直方图上选择一个范围。单击要开始选择的位置，然后从左向右拖动，突出显示一个要包括在跟踪筛选条件中的延迟范围。



选择范围后，您可以选择 Zoom，只查看该部分的直方图并细化您的选择。



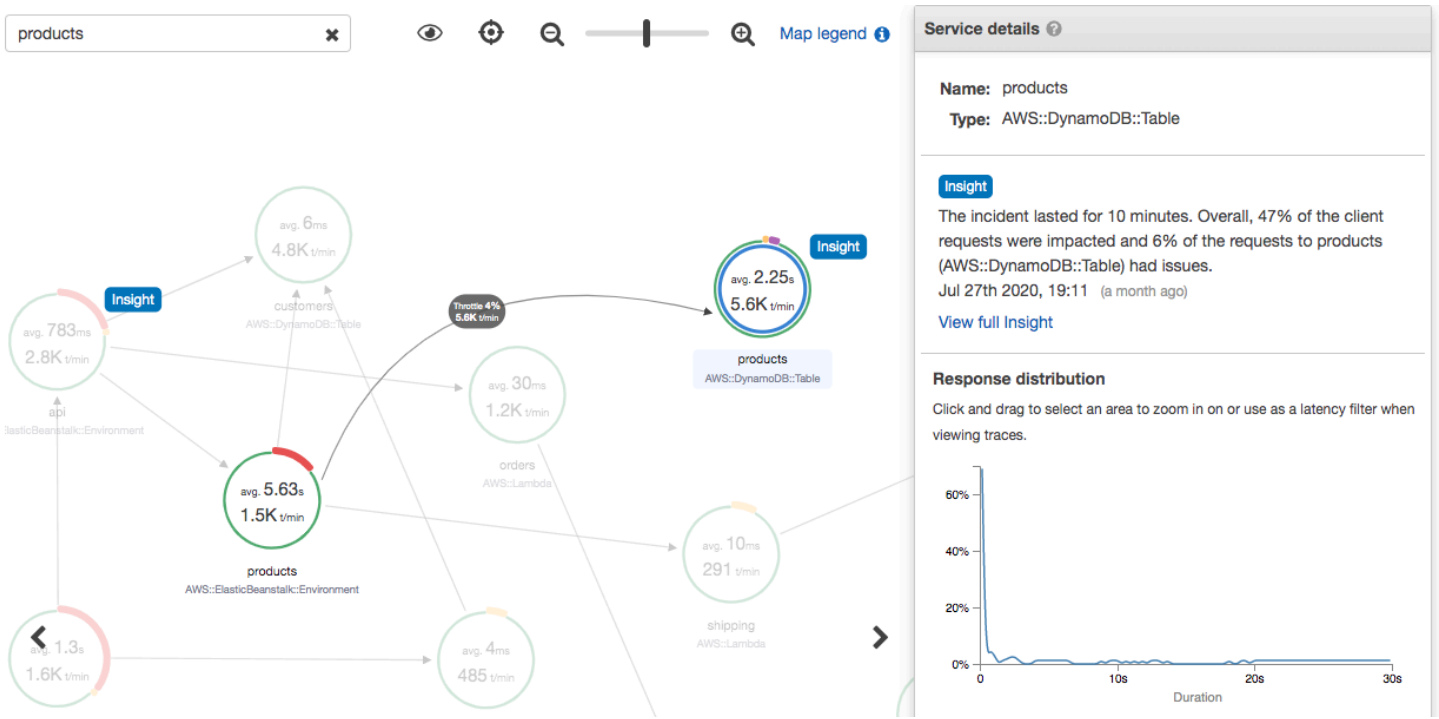
将焦点设置为感兴趣的区域后，选择 View traces。

使用 X-Ray 见解

AWS X-Ray 持续分析您账户中的跟踪数据，以识别应用程序中的紧急问题。当错误率超出预期范围时，它会创建见解来记录问题并跟踪问题产生的影响，直到问题被解决。借助见解，您可以：

- 确定应用程序哪里出现了问题，问题的根本原因，以及关联的影响。您可以从见解提供的影响分析中获取到某个问题的严重性和优先级。
- 随着时间推移，当问题发生变化时收到通知。Insights 通知可以通过 Amazon EventBridge 与您的监控和警报解决方案集成。这种集成让您可以根据问题的严重性发送自动化电子邮件或警报。

X-Ray 控制台可识别跟踪地图中正在发生事件的节点。若要查看见解摘要，请选择受影响的节点。还可以从左侧导航窗格中选择见解，查看和筛选见解。



当 X-Ray 在服务地图的一个或多个节点中检测到异常时，便会创建见解。该服务使用统计建模来预测应用程序中服务的预期故障率。在前面的示例中，异常是故障的 AWS Elastic Beanstalk增加。Elastic Beanstalk 服务器经历了多次 API 调用超时，导致下游节点出现异常。

在 X-Ray 控制台中启用见解

必须为希望使用见解功能的每个组都启用见解。可以从组页面启用见解。

1. 打开 X-Ray 控制台。
2. 选择现有组或通过选择创建组创建一个新组，然后选择启用见解。有关如何在 X-Ray 控制台中配置组的更多信息，请参阅配置组。
3. 在左侧导航窗格中选择见解，然后选择要查看的见解。

> **Note**
>
> X-Ray 使用 GetInsightSummaries GetInsight GetInsightEvents、、和 GetInsightImpactGraph API 操作从见解中检索数据。要查看见解，请使用 AWSXray ReadOnlyAccess IAM 托管策略或将以下自定义策略添加到您的 IAM 角色：

```
{
   "Version": "2012-10-17",
   "Statement": [
       {
           "Effect": "Allow",
           "Action": [
               "xray:GetInsightSummaries",
               "xray:GetInsight",
               "xray:GetInsightEvents",
               "xray:GetInsightImpactGraph"
           ],
           "Resource": [
               "*"
           ]
       }
   ]
}
```

有关更多信息，请参阅 如何 AWS X-Ray 与 IAM 配合使用。

启用见解通知

可以使用见解通知为每个见解事件创建一则通知，例如，当创建见解、发生重大更改，或关闭见解时。客户可以通过 Amazon EventBridge 事件接收这些通知，并使用条件规则采取诸如 SNS 通知、Lambda 调用、向 SQS 队列发布消息或任何目标支持的操作。 EventBridge 我们会尽可能发送见解通知，但并不保证。有关目标的更多信息，请参阅 Amazon EventBridge 目标。

您可以从组页面为任意已启用见解的组启用见解通知。

为 X-Ray 组启用通知

1. 打开 X-Ray 控制台。

2. 选择现有组或通过选择创建组创建一个新组，确保选中启用见解，然后选择启用通知。有关如何在 X-Ray 控制台中配置组的更多信息，请参阅配置组。

配置 Amazon EventBridge 条件规则

1. 打开亚马逊 EventBridge 控制台。
2. 导航到左侧导航栏中的规则，然后选择创建规则。
3. 提供规则的名称和描述。
4. 选择事件模式，然后选择自定义模式。提供一个包含 `"source": [ "aws.xray" ]` 和 `"detail-type": [ "AWS X-Ray Insight Update" ]` 的模式。以下是可能的一些示例模式。
   - 事件模式要匹配 X-Ray 见解中的所有传入事件：

     ```
     {
     "source": [ "aws.xray" ],
     "detail-type": [ "AWS X-Ray Insight Update" ]
     }
     ```

   - 事件模式要匹配指定的 **`state`** 和 **`category`**：

     ```
     {
     "source": [ "aws.xray" ],
     "detail-type": [ "AWS X-Ray Insight Update" ],
     "detail": {
            "State": [ "ACTIVE" ],
            "Category": [ "FAULT" ]
       }
     }
     ```

5. 选择并配置当某个事件匹配此规则时，您想要调用的目标。
6. （可选）提供标签以便更轻松地识别和选择此规则。
7. 选择创建。

> **Note**
>
> X-Ray insights 通知会将事件发送到亚马逊 EventBridge，而亚马逊目前不支持客户托管密钥。有关更多信息，请参阅 中的数据保护 AWS X-Ray。

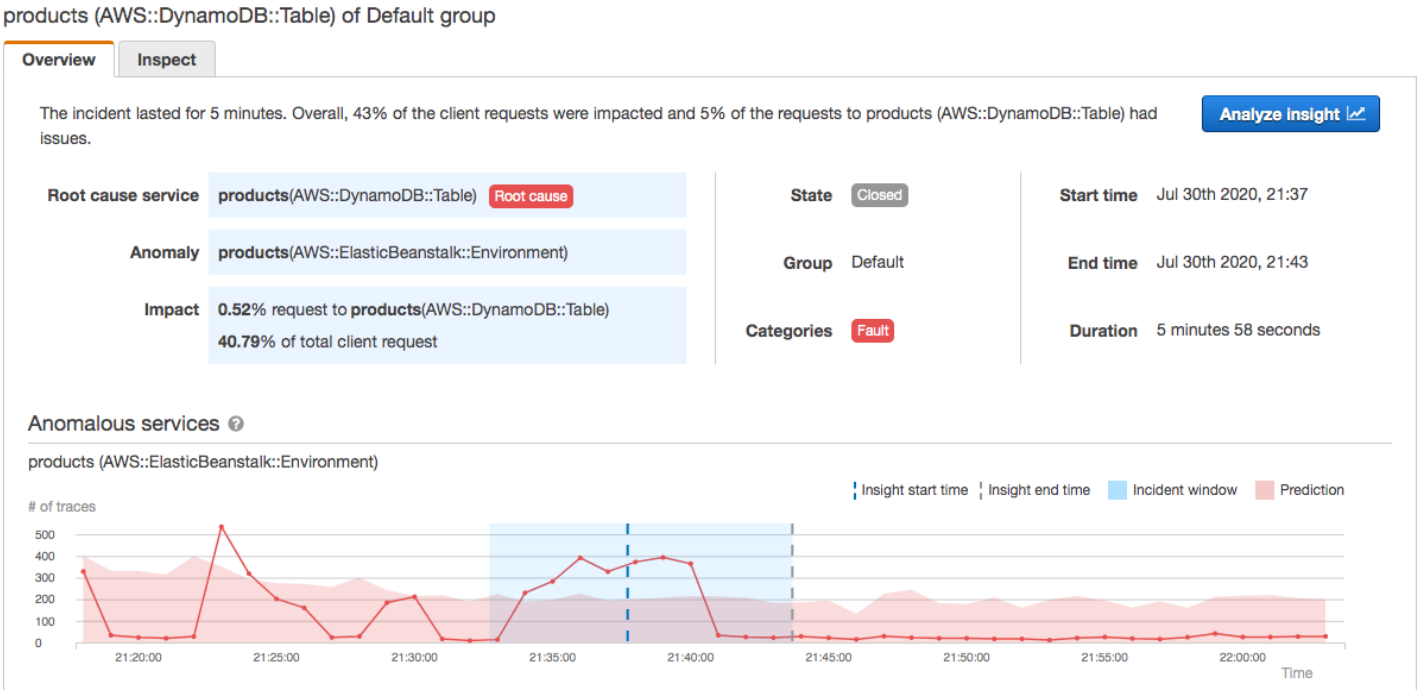
见解

见解概述页面试图回答以下三个关键问题：

- 什么是潜在问题？
- 什么是根本原因？
- 什么是影响？

异常服务一节介绍了每个服务的时间表，展示了事件过程中故障率的变化情况。时间表显示了出现故障的跟踪数量，根据记录的流量，以实心条带指明预计的故障数量。事件窗口将见解的持续时间可视化。当 X-Ray 观察到指标出现异常并且启用见解后依旧存在的时候，事件窗口启动。

以下示例显示的是导致某个事件的故障出现增加：

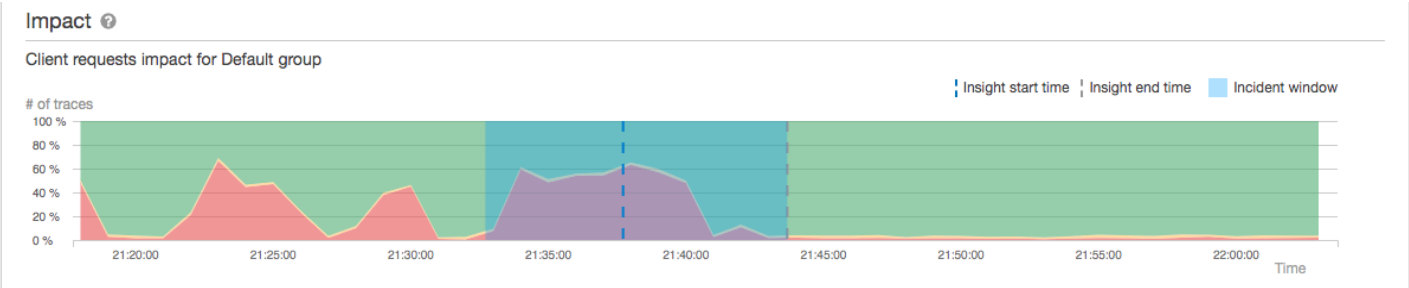


根本原因一节显示了聚焦根本原因服务和受影响路径的跟踪地图。可以选择根本原因地图右上角的眼睛图标，以隐藏未受影响的节点。根本原因服务是 X-Ray 识别到异常的最远的下游节点。它可以代表您

检测的某项服务，或是您服务使用检测客户端调用过的外部服务。例如，如果您使用 AWS 经过检测的软件开发工具包客户端调用 Amazon DynamoDB，那么 DynamoDB 错误的增加会导致以 DynamoDB 为根本原因的见解。

若要进一步调查根本原因，请在根本原因图上选择查看根本原因详情。您可以使用分析页面来调查根本原因以及相关消息。有关更多信息，请参阅 与 Analytics 控制台交互。

在地图中继续上游的故障会影响多个节点，并会导致多种异常。如果某个故障一直传回到发出请求的用户，就会出现客户端故障。这是跟踪地图的根节点的一个故障。影响示意图为整个组的客户端体验提供了时间表。根据以下状态的比例来计算此体验：故障、错误、瓶颈和没错。



此示例显示在事件发生过程中，在顶部节点带有故障的跟踪增加了。下游服务中的事件并不总是与客户端错误的增加相对应。

选择分析见解会在窗口中打开 X-Ray Analytics 控制台，您可以在其中深入研究产生见解的跟踪集。有关更多信息，请参阅 与 Analytics 控制台交互。

了解影响

AWS X-Ray 衡量持续存在的问题所造成的影响，作为生成见解和通知的一部分。有两种方法来衡量影响：

- 对 X-Ray 组的影响

- 对根本原因服务的影响

此影响由在给定时间内发生故障或造成错误的请求比例决定。通过这种影响分析，您可以根据自己的特定情况得出问题的严重性和优先级。除了见解通知以外，影响还是控制台体验的一部分。

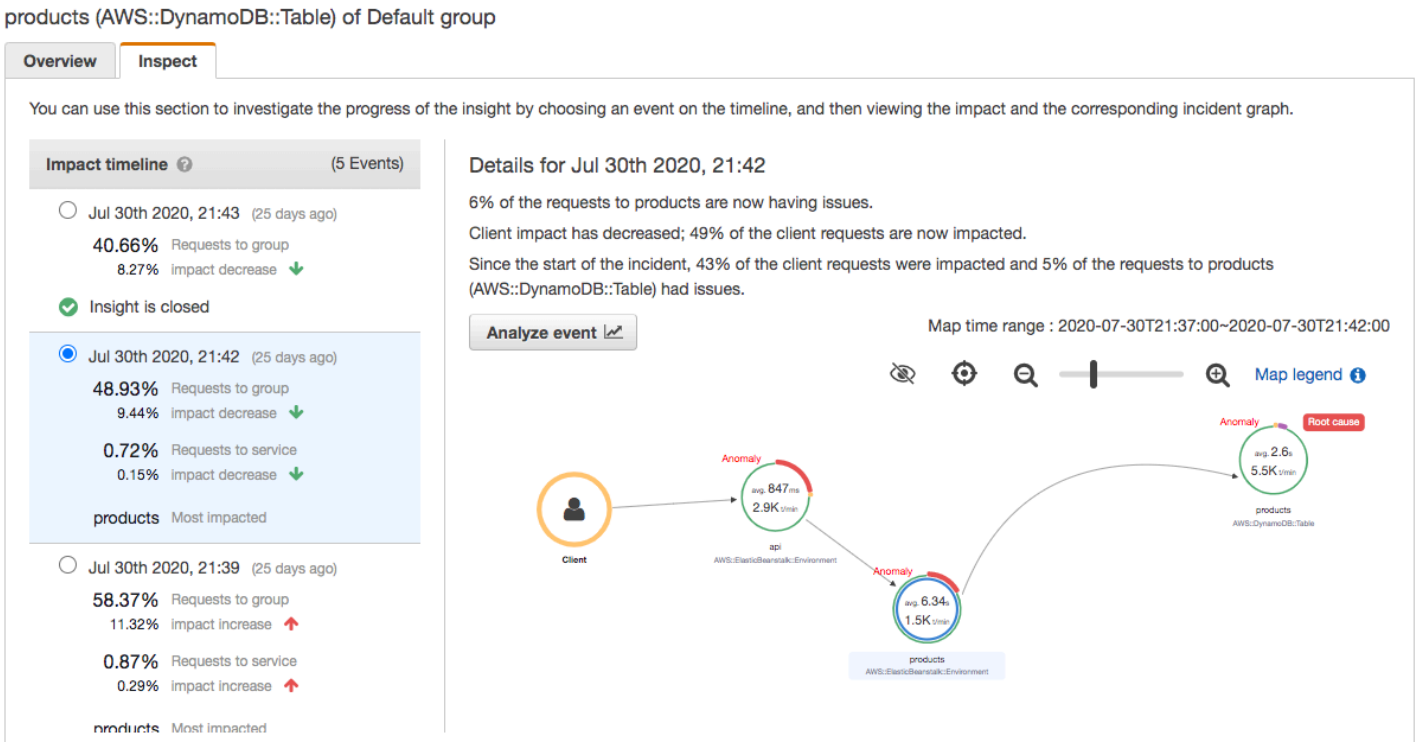
重复数据删除

AWS X-Ray 见解可以消除多个微服务中的重复问题。它使用异常检测来确定是某个问题根本原因的服务，确定其他相关服务是否由于同一根本原因展现出异常行为，并将结果记录为单个见解。

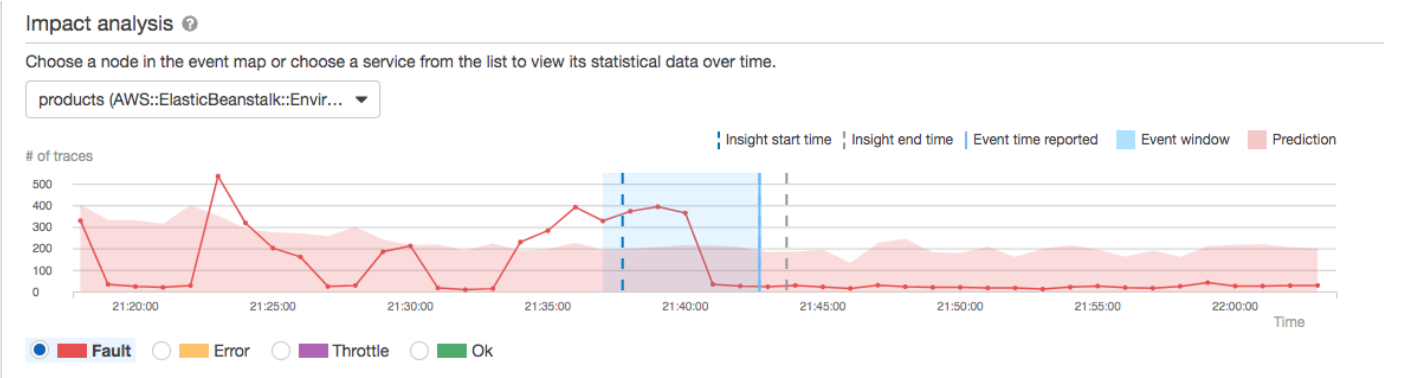
查看见解的进展

X-Ray 会定期重新评估见解，直到它们得到解决，并将每项明显的中间更改记录为通知，该通知可以作为 Ama EventBridge zon 事件发送。这使您能够构建流程和工作流程以确定问题如何随时间推移而发生变化，并采取适当的措施，例如发送电子邮件或使用 EventBridge与警报系统集成。

您可以在影响页面上的影响时间表中查看事件活动。默认情况下，时间表会显示最受影响的服务，直到您选择另一项服务。



要查看某个事件的跟踪地图和示意图，请从影响时间表中选择该事件。跟踪地图显示应用程序中受事件影响的服务。在“影响分析”下，图表显示选定节点和组中客户端的故障时间表。



若要更加深入地了解某个事件中涉及的跟踪，请在检查页面上选择分析事件。可以使用分析页面来优化跟踪列表并确定受影响的用户。有关更多信息，请参阅 与 Analytics 控制台交互。

与 Analytics 控制台交互

AWS X-Ray Analytics 控制台是一个交互式工具，用于解释跟踪数据，从而快速了解您的应用程序及其底层服务的性能。借助该控制台，您可以通过交互式响应时间图表和时间序列图表探索、分析和直观地显示跟踪。

当在 Analytics 控制台中进行选择时，控制台构造筛选条件以反映所有跟踪的所选子集。您可以通过单击与当前跟踪集关联的指标和字段的图表和面板，使用越来越精细的筛选条件细化活动的数据集。

主题

控制台功能

X-Ray Analytics 使用以下关键功能对跟踪数据进行分组、筛选、比较和量化。

特征

特征	描述
组	初始选定的组为 Default。要更改检索的组，请从主要筛选表达式搜索栏右侧的菜单中选择不同的组。要了解有关组的更多信息，请参阅将筛选表达式与组结合使用。
已检索跟踪	默认情况下，Analytics 控制台根据所选组中的所有跟踪生成图表。检索的跟踪表示您工作集中的所有跟踪。您可以在此平铺中找到跟踪计数。您应用于主搜索栏的筛选表达式可以细化和更新检索的跟踪。
显示在图表中/从图表中隐藏	一个开关，用于对照检索的跟踪比较活动的组。要对照任何活动的筛选条件比较与组相关的数据，请选择显示在图表中。要从图表中删除此视图，请选择从图表中隐藏。
已筛选跟踪集 A	通过与图表和表进行交互，应用筛选条件来创建筛选跟踪集 A 的条件。应用筛选条件后，可以在此平铺内计算适用跟踪的数量以及跟踪占所检索总数的百分比。筛选条件作为标签填充到筛选跟踪集 A 磁贴中，也可以将其从此磁贴中删除。
细化	此函数根据应用到跟踪集 A 的筛选条件更新已检索的跟踪集。细化已检索的跟踪集会根据跟踪集 A 的筛选条件刷新所有已检索的跟踪的工作集。已检索的跟踪的工作集是组中所有跟踪的采样子集。
筛选跟踪集 B	创建后，筛选跟踪集 B 是筛选跟踪集 A 的副本。若要比较这两个跟踪集，请选择将会应用于跟踪集 B 的筛选，而跟踪集 A 则保持不变。应用筛选条件后，可以在此平铺内计算适用跟踪的数量以及跟踪占所检索总数的百分比。筛选条件

特征	描述
	作为标签填充到已筛选跟踪集 B 磁贴中，也可以将其从此磁贴中删除。
响应时间根本原因实体路径	记录的实体路径表。X-Ray 确定跟踪中的哪个路径是响应时间的最可能原因。格式指示所遇到的实体的层次结构，结尾是响应时间根本原因。使用这些行来筛选周期性的响应时间故障。有关通过 API 自定义根本原因筛选条件和获取数据的更多信息，请参阅检索和细化根本原因分析。
增量 (Δ)	在跟踪集 A 和 B 都处于活动状态时添加到指标表中的列。增量列计算跟踪集 A 和跟踪集 B 之间的跟踪百分比差异。

响应时间分配

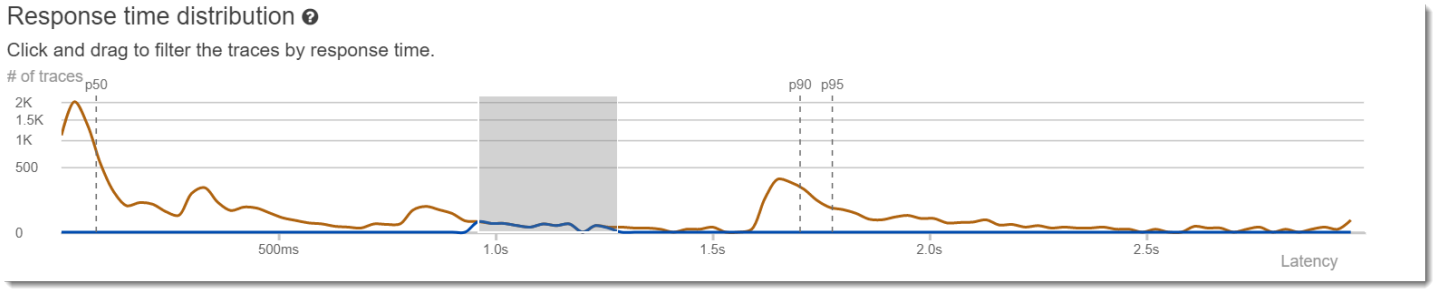
X-Ray Analytics 控制台生成两个主要图表以帮助您直观显示跟踪：响应时间分配和时间序列活动。本节和下面的内容提供有关每个图表的示例，并说明有关如何阅读这些图表的基本知识。

以下是与响应时间线状图关联的颜色（时间序列图使用相同的颜色方案）：

- 组中的所有跟踪 - 灰色
- 已检索跟踪 - 橙色
- 已筛选跟踪集 A - 绿色
- 已筛选跟踪集 B - 蓝色

Example - 响应时间分配

响应时间分配是一个图表，用于显示给定响应时间内的跟踪数量。单击并拖动以在响应时间分配内进行选择。这会针对特定响应时间内的所有跟踪的工作跟踪集，选择和创建一个名为 `responseTime` 的筛选条件。

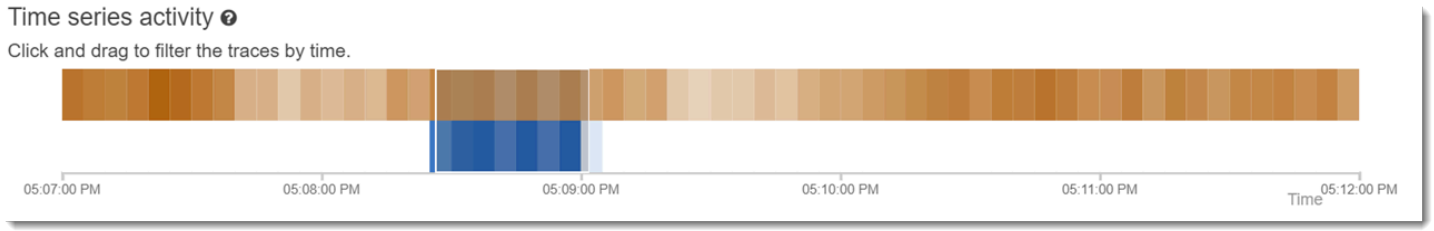


时间序列活动

时间序列活动图表显示给定时间段的跟踪数量。颜色指示反映响应时间分配的线图颜色。活动序列中的颜色块越深越饱满，则表示给定时间的跟踪数越多。

Example - 时间序列活动

单击并拖动以在时间序列活动图表内进行选择。这会针对特定时间范围内的所有跟踪的工作跟踪集，选择和创建一个名为 `timerange` 的筛选条件。



工作流程示例

下面的示例说明 X-Ray Analytics 控制台的常见使用案例。每个示例演示控制台体验的一个关键功能。这些示例作为一个组遵循基本的疑难排查工作流。这些步骤介绍了如何先发现运行状况不佳的节点，以及如何与 Analytics 控制台交互以自动生成比较查询。通过查询缩小范围后，您就可以查看感兴趣的跟踪的详细信息，以确定是什么问题影响了服务的健康运行。

在服务图表上观察故障

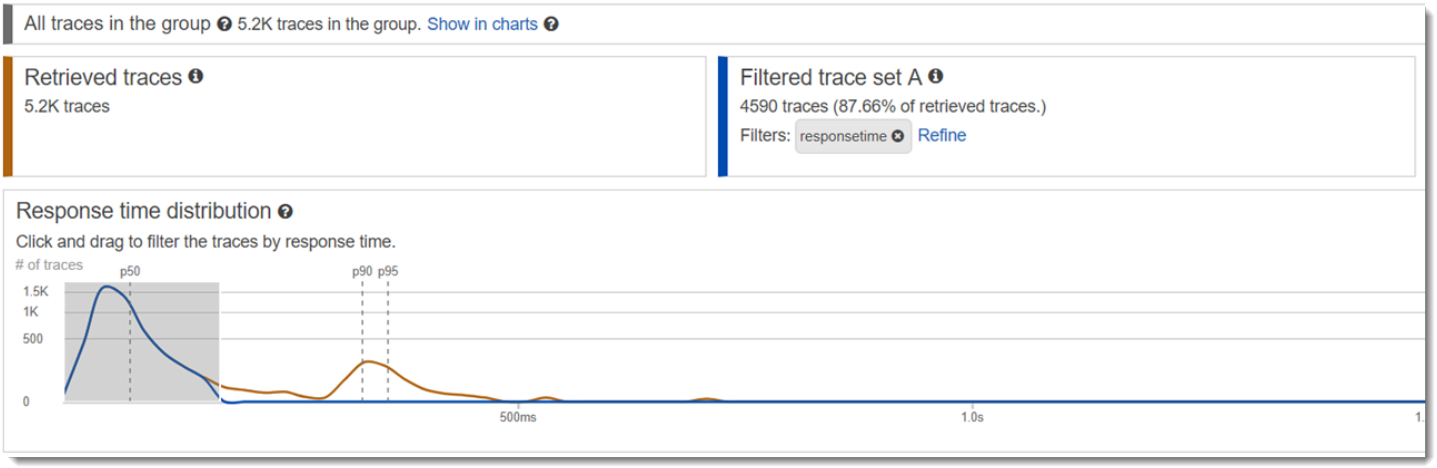
跟踪地图根据成功调用与错误和故障的比率为每个节点配置颜色，从而指示这些节点的运行状况。当您在节点上看到红色百分比时，这指示一个故障。使用 X-Ray Analytics 控制台调查此故障。

有关如何解读跟踪地图的更多信息，请参阅查看跟踪地图。

确定高峰响应时间

使用响应时间分配，您可以观察高峰响应时间。通过选择高峰响应时间，图表下方的各个表格将会更新，以显示所有关联的指标，如状态代码。

您可以通过单击并拖动 X-Ray 来选择和创建筛选条件。筛选条件会以灰色阴影的形式显示在线状图顶部。现在，您可以沿着分配左右拖动阴影区以更新您的选择和筛选条件。



查看所有标有状态代码的跟踪

您可以使用图表下方的指标表钻取所选峰值内的跟踪。通过单击HTTP 状态代码表中的行，您可以自动对工作数据集创建筛选条件。例如，您可以查看状态代码为 500 的所有跟踪。这会在跟踪集平铺中创建一个名为 `http.status` 的筛选条件标签。

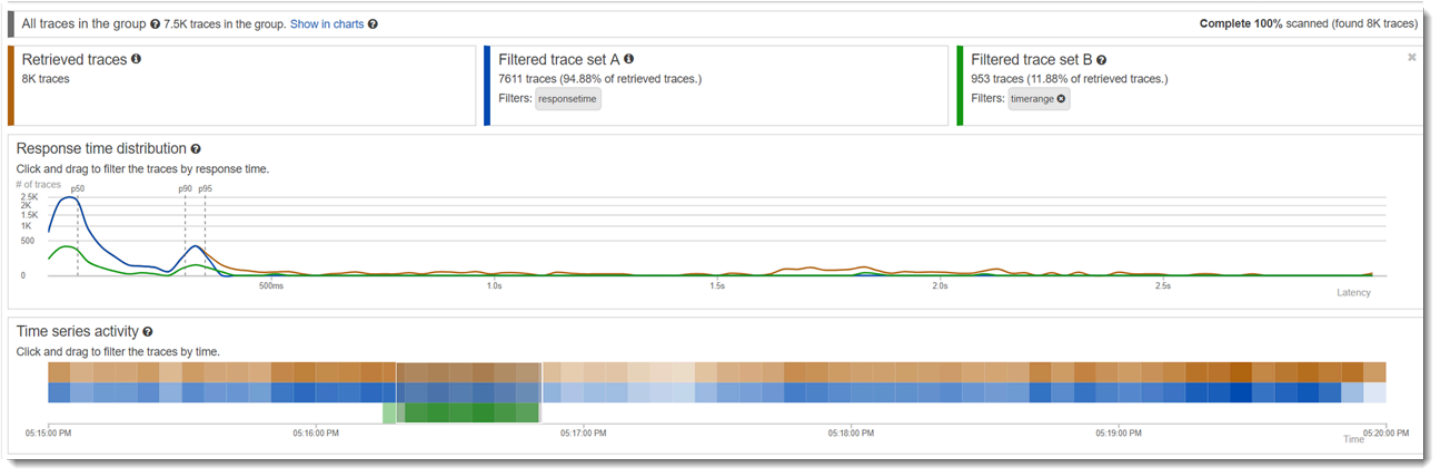
查看子组中与用户关联的所有项目

根据用户、URL、响应时间根本原因或其他预定义的属性钻取错误集。例如，要额外筛选状态代码为 500 的跟踪集，请从 USERS 表中选择一行。这会导致在跟踪集平铺中产生两个筛选条件标签：`http.status`（如前面指定）和 `user`。

比较具有不同标准的两组跟踪

跨不同用户及其 POST 请求进行比较，以查找其他差异和关联。应用您的第一个筛选条件集。它们通过响应时间分配中的蓝线定义。然后，选择比较。最初，这会对跟踪集 A 创建筛选器的副本。

要继续，请定义要应用于跟踪集 B 的新的筛选条件集。这第二个集合由绿线表示。以下示例根据蓝色和绿色颜色方案显示不同的行。



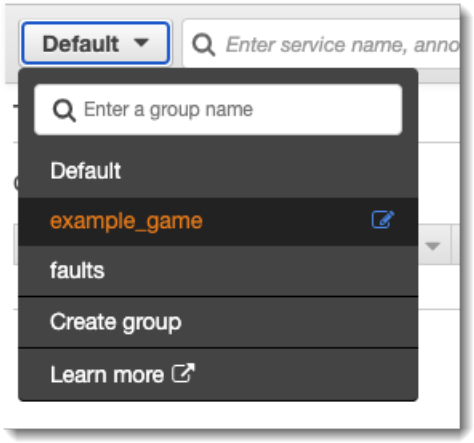
确定感兴趣的跟踪并查看其详细信息

当您使用控制台筛选条件缩小范围时，指标表下方的跟踪列表将变得更有意义。跟踪列表将有关 URL、用户和状态代码的信息合并显示在一个视图中。如需更多见解，请从此表中选择一行，以打开跟踪的详细信息页面并查看其时间线和原始数据。

配置组

组是由筛选条件表达式定义的跟踪的集合。您可以使用群组来生成其他服务图表并提供 Amazon CloudWatch 指标。您可以使用 AWS X-Ray 控制台或 X-Ray API 为服务创建组并进行管理。本主题介绍如何使用 X-Ray 控制台来创建和管理组。请参阅组，了解如何使用 X-Ray API 管理组。

您可以为跟踪地图、跟踪或分析创建跟踪组。创建组后，该组会在以下全部三个页面上的组下拉菜单中变为一个可用的筛选条件：跟踪地图、跟踪和分析。



组由其名称或 Amazon 资源名称 (ARN) 标识，并包含筛选条件表达式。此服务将比较传入到表达式的跟踪并相应地存储它们。请参阅使用筛选条件表达式，详细了解如何构建筛选表达式。

更新组的筛选条件表达式不会更改已记录的数据。更新仅应用于后续跟踪。这可能会生成新旧表达式的合并图。为避免发生这种情况，请删除当前群组并创建一个新的组。

> **Note**
>
> 群组按检索到的符合筛选条件表达式的追踪数量计费。有关更多信息，请参阅 AWS X-Ray 定价。

主题

创建组

> **Note**
>
> 现在，您可以从亚马逊 CloudWatch 控制台中配置 X-Ray 组。也可以继续使用 X-Ray 控制台。

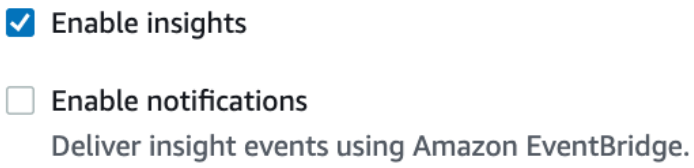
CloudWatch console

1. 登录 AWS Management Console 并打开 CloudWatch 控制台，网址为https://console.aws.amazon.com/cloudwatch/。
2. 在左侧导航窗格中，选择设置。
3. 在 X-Ray 跟踪部分中的组下，选择查看设置。
4. 在组列表上，选择创建组。
5. 在创建组页面上，输入组的名称。组名称最多可包含 32 个字符，可包含字母数字字符和破折号。组名称区分大小写。

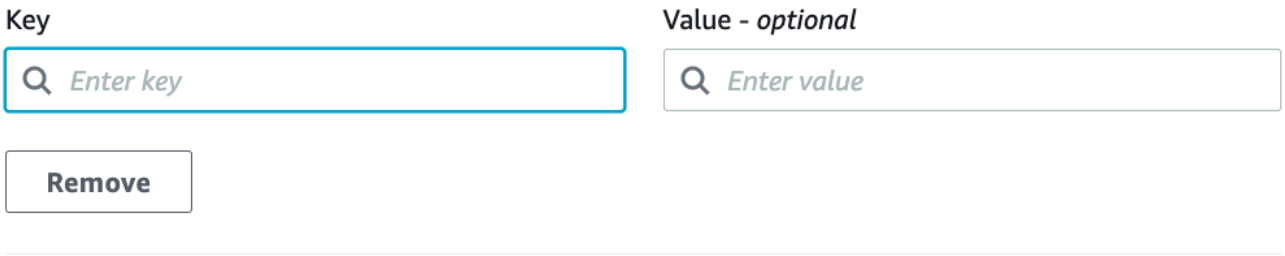
6. 输入筛选条件表达式。请参阅使用筛选条件表达式，详细了解如何构建筛选表达式。在以下示例中，组筛选服务 `api.example.com` 中的错误跟踪以及发送到该服务的请求，其中呼应时间大于或等于 5 秒的情况。

```
fault = true AND http.url CONTAINS "example/game" AND responsetime >= 5
```

7. 在见解中，启用或禁用组的见解访问。有关见解的更多信息，请参阅 使用 X-Ray 见解。



8. 在标签中，选择添加新标签以输入标签键，以及选填的标签值。根据需要继续添加其他标签。标签键必须是唯一的。要删除标签，请选择每个标签下方的删除。有关标签的更多信息，请参阅 标记 X-Ray 采样规则和组。



9. 选择创建群组。

X-Ray console

1. 登录 AWS Management Console 并在https://console.aws.amazon.com/xray/家中打开 X-Ray 控制台。
2. 在左侧导航窗格中的组页面中，或是从以下任意页面中的组菜单中，打开创建组页面：跟踪地图、跟踪和分析。
3. 在创建组页面上，输入组的名称。组名称最多可包含 32 个字符，可包含字母数字字符和破折号。组名称区分大小写。
4. 输入筛选条件表达式。请参阅使用筛选条件表达式，详细了解如何构建筛选表达式。在以下示例中，组筛选服务 `api.example.com` 中的错误跟踪以及发送到该服务的请求，其中呼应时间大于或等于 5 秒的情况。

```
fault = true AND http.url CONTAINS "example/game" AND responsetime >= 5
```

5. 在见解中，启用或禁用组的见解访问。有关见解的更多信息，请参阅 使用 X-Ray 见解。

6. 在标签中，输入一个标签键和可选的标签值。添加标签时会出现一个新行，供您添加另一个标签。标签键必须是唯一的。若要删除标签，请选择标签行末的 X。有关标签的更多信息，请参阅 标记 X-Ray 采样规则和组。

7. 选择创建群组。

应用组

CloudWatch console

1. 登录 AWS Management Console 并打开 CloudWatch 控制台，网址为https://console.aws.amazon.com/cloudwatch/。
2. 从 X-Ray 跟踪下的导航窗格中打开以下任意一个页面：
   - 跟踪地图
   - 跟踪
3. 在按 X-Ray 组筛选筛选条件中输入组名称。页面上显示的数据会发生变化，以匹配组中设置的筛选条件表达式。

X-Ray console

1. 登录 AWS Management Console 并在https://console.aws.amazon.com/xray/家中打开 X-Ray 控制台。
2. 从导航窗格中打开以下任意一个页面：
   - 跟踪地图
   - 跟踪

- 分析

3. 在组菜单中，选择在 the section called “创建组” 中创建的组。页面上显示的数据会发生变化，以匹配组中设置的筛选条件表达式。

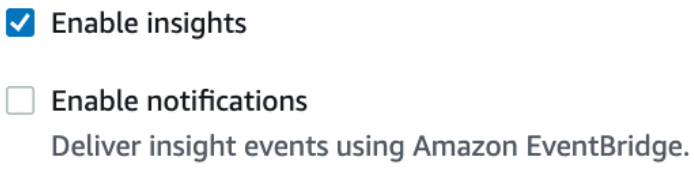
编辑组

CloudWatch console

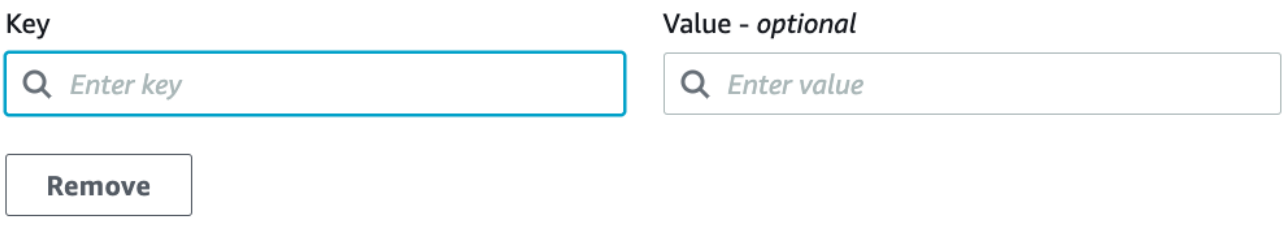
1. 登录 AWS Management Console 并打开 CloudWatch 控制台，网址为https://console.aws.amazon.com/cloudwatch/。
2. 在左侧导航窗格中，选择设置。
3. 在 X-Ray 跟踪部分中的组下，选择查看设置。
4. 从组部分中选择一个组，然后选择编辑。
5. 尽管您无法重命名组，但可以更新筛选条件表达式。请参阅使用筛选条件表达式，详细了解如何构建筛选表达式。在以下示例中，组筛选服务 `api.example.com` 中的错误跟踪，其中请求 URL 地址包含 `example/game`，以及呼应时间大于或等于 5 秒的情况。

```
fault = true AND http.url CONTAINS "example/game" AND responsetime >= 5
```

6. 在见解中，启用或禁用组的见解访问。有关见解的更多信息，请参阅 使用 X-Ray 见解。



7. 在标签中，选择添加新标签以输入标签键，以及选填的标签值。根据需要继续添加其他标签。标签键必须是唯一的。要删除标签，请选择每个标签下方的删除。有关标签的更多信息，请参阅 标记 X-Ray 采样规则和组。



8. 更新完组后，选择更新组。

X-Ray console

1. 登录 AWS Management Console 并在https://console.aws.amazon.com/xray/家中打开 X-Ray 控制台。
2. 执行以下其中一个操作以打开编辑组页面。

   a. 在群页面上，选择某个组的名称进行编辑。

   b. 在以下任意页面之一上的组菜单上，指向某个组，然后选择编辑。

   - 跟踪地图
   - 跟踪
   - 分析

3. 尽管您无法重命名组，但可以更新筛选条件表达式。请参阅使用筛选条件表达式，详细了解如何构建筛选表达式。在以下示例中，组筛选服务 `api.example.com` 中的错误跟踪，其中请求 URL 地址包含 `example/game`，以及呼应时间大于或等于 5 秒的情况。

   ```
   fault = true AND http.url CONTAINS "example/game" AND responsetime >= 5
   ```

4. 在见解中，启用或禁用组的见解和见解通知。有关见解的更多信息，请参阅 使用 X-Ray 见解。

5. 在标签中，编辑标签键和值。标签键必须是唯一的。标签值是选填，如果需要可以删除。若要删除标签，请选择标签行末的 X。有关标签的更多信息，请参阅 标记 X-Ray 采样规则和组。

6. 更新完组后，选择更新组。

克隆组

克隆组会创建具有现有组的筛选条件表达式和标签的新组。克隆组时，新组具有被克隆组的同一名称，名称后附加 `-clone`。

CloudWatch console

1. 登录 AWS Management Console 并打开 CloudWatch 控制台，网址为https://console.aws.amazon.com/cloudwatch/。
2. 在左侧导航窗格中，选择设置。
3. 在 X-Ray 跟踪部分中的组下，选择查看设置。
4. 从组部分中选择一个组，然后选择克隆。
5. 在创建群组页面上，群组的名称为*group-name*-clone。（可选）输入组的新名称。组名称最多可包含 32 个字符，可包含字母数字字符和破折号。组名称区分大小写。
6. 您可以保留现有组中的筛选条件表达式，也可以选择输入新的筛选条件表达式。请参阅使用筛选条件表达式，详细了解如何构建筛选表达式。在以下示例中，组筛选服务 `api.example.com` 中的错误跟踪以及发送到该服务的请求，其中呼应时间大于或等于 5 秒的情况。

   ```
   service("api.example.com") { fault = true OR responsetime >= 5 }
   ```

7. 如果需要，可在标签中编辑标签键和值。标签键必须是唯一的。标签值是选填，如果需要可以删除。若要删除标签，请选择标签行末的 X。有关标签的更多信息，请参阅 标记 X-Ray 采样规则和组。
8. 选择创建群组。

X-Ray console

1. 登录 AWS Management Console 并在https://console.aws.amazon.com/xray/家中打开 X-Ray 控制台。
2. 从左侧导航窗格中打开组页面，然后选择想要克隆的组的名称。
3. 从操作菜单中选择克隆组。
4. 在创建群组页面上，群组的名称为*group-name*-clone。（可选）输入组的新名称。组名称最多可包含 32 个字符，可包含字母数字字符和破折号。组名称区分大小写。
5. 您可以保留现有组中的筛选条件表达式，也可以选择输入新的筛选条件表达式。请参阅使用筛选条件表达式，详细了解如何构建筛选表达式。在以下示例中，组筛选服务 `api.example.com` 中的错误跟踪以及发送到该服务的请求，其中呼应时间大于或等于 5 秒的情况。

   ```
   service("api.example.com") { fault = true OR responsetime >= 5 }
   ```

6. 如果需要，可在标签中编辑标签键和值。标签键必须是唯一的。标签值是选填，如果需要可以删除。若要删除标签，请选择标签行末的 X。有关标签的更多信息，请参阅 标记 X-Ray 采样规则和组。
7. 选择创建群组。

删除组

按照本节中的步骤删除组。您不能删除默认组。

CloudWatch console

1. 登录 AWS Management Console 并打开 CloudWatch 控制台，网址为https://console.aws.amazon.com/cloudwatch/。
2. 在左侧导航窗格中，选择设置。
3. 在 X-Ray 跟踪部分中的组下，选择查看设置。
4. 从组部分中选择一个组，然后选择删除。
5. 请在提示您进行确认时选择删除。

X-Ray console

1. 登录 AWS Management Console 并在https://console.aws.amazon.com/xray/家中打开 X-Ray 控制台。
2. 从左侧导航窗格中打开组页面，然后选择想要删除的组的名称。
3. 在操作菜单上，选择删除组。
4. 请在提示您进行确认时选择删除。

在 Amazon 中查看群组指标 CloudWatch

创建组后，将根据组的筛选条件表达式检查传入跟踪，因为它们存储在 X-Ray 服务中。每分钟向 Amazon CloudWatch 发布符合每个标准的追踪数量的指标。在 “编辑群组” 页上选择 “查看指标”，即可打开 CloudWatch控制台，进入 “指标” 页面。有关如何使用 CloudWatch 指标的更多信息，请参阅亚马逊 CloudWatch 用户指南中的使用亚马逊 CloudWatch指标。

CloudWatch console

1. 登录 AWS Management Console 并打开 CloudWatch 控制台，网址为https://console.aws.amazon.com/cloudwatch/。
2. 在左侧导航窗格中，选择设置。
3. 在 X-Ray 跟踪部分中的组下，选择查看设置。
4. 从组部分中选择一个组，然后选择编辑。
5. 在编辑组页面上，选择查看指标。

   CloudWatch 控制台 “指标” 页面将在新选项卡中打开。

X-Ray console

1. 登录 AWS Management Console 并在https://console.aws.amazon.com/xray/家中打开 X-Ray 控制台。
2. 从左侧导航窗格中打开组页面，然后选择想要查看其指标的组的名称。
3. 在编辑组页面上，选择查看指标。

   CloudWatch 控制台 “指标” 页面将在新选项卡中打开。

配置采样规则

您可以使用 AWS X-Ray 控制台为您的服务配置采样规则。支持带采样配置的主动跟踪 AWS 服务的 X-Ray SDK 使用采样规则来确定要记录哪些请求。

主题

配置采样规则

您可以为以下使用案例配置采样：

- API 网关加密 - API 网关支持采样和活动跟踪。要在 API 阶段启用活动跟踪，请参阅Amazon API Gateway 主动追踪支持 AWS X-Ray。
- AWS AppSync— AWS AppSync 支持采样和主动跟踪。要启用对 AWS AppSync 请求的主动跟踪，请参阅使用 AWS X-Ray 进行跟踪。
- 计算平台上的 Instrument X-Ray SDK — 当使用诸如亚马逊 EC2、Amazon ECS 或之类的计算平台时 AWS Elastic Beanstalk，如果应用程序已使用最新的 X-Ray SDK 进行检测，则支持采样。

自定义抽样规则

您可以通过自定义采样规则来控制记录的数据量。也可以修改采样行为，而无需修改或重新部署代码。采样规则向 X-Ray SDK 告知要为一系列条件记录的请求数。默认情况下，X-Ray 开发工具包每秒记录第一个请求，以及任何其他请求的百分之五。每秒一个请求是容器。这可确保只要服务正在处理请求，就会每秒至少记录一个跟踪。5% 是对超出容器尺寸的额外请求进行采样的比率。

您可以将 X-Ray SDK 配置为从您包含在代码中的 JSON 文档读取采样规则。但是，当您运行服务的多个实例时，每个实例都会单独执行采样。这会导致采样的请求的总体比例升高，因为所有实例的容器都会被有效地一起添加。此外，要更新本地采样规则，则必须重新部署您的代码。

通过在 X-Ray 控制台中定义采样规则，然后配置 SDK 以从 X-Ray 服务读取规则，您可以避免这两个问题。该服务将管理每条规则的容器，并向您的服务的每个实例分配配额以基于正在运行的实例数均匀地分配容器。容器限制是根据您设置的规则计算的。由于规则是在服务中配置的，您可以管理规则而不进行额外的部署。

> **Note**
>
> X-Ray 会尽力应用采样规则，在某些情况下，有效采样率可能并不与配置的采样规则完全匹配。但是，随着时间推移，采样的请求数量应接近配置的百分比。

现在，您可以在亚马逊 CloudWatch 控制台中配置 X-Ray 采样规则。也可以继续使用 X-Ray 控制台。

CloudWatch console

在 CloudWatch 控制台中配置采样规则

1. 登录 AWS Management Console 并打开 CloudWatch 控制台，网址为https://console.aws.amazon.com/cloudwatch/。
2. 在左侧导航窗格中，选择设置。
3. 在 X-Ray 跟踪部分中的采样规则下，选择查看设置。
4. 要创建规则，请选择创建采样规则。

   要编辑规则，请选择该规则，然后选择编辑即可进行编辑。

   要删除规则，请选择该规则，然后选择删除即可将其删除。

X-Ray console

在 X-Ray 控制台中配置采样规则

1. 打开 X-Ray 控制台。
2. 在左侧导航窗格中，选择采样。
3. 要创建规则，请选择创建采样规则。

   要编辑规则，请选择规则的名称。

   要删除规则，请选择一条规则并使用操作菜单来删除它。

采样规则选项

以下选项可用于每条规则。字符串值可以使用通配符来匹配单个字符 (?) 或零或多个字符 (*)。

采样规则选项

- 规则名称（字符串）— 一个唯一的规则名称。
- 优先级（1 和 9999 之间的整数）— 采样规则的优先级。服务按优先级的上升顺序评估规则，并与匹配的第一条规则进行抽样决策。
- 容器（非负整数）- 在应用固定比率之前，每秒与检测匹配的固定请求数。该容器不由服务直接使用，但适用于所有使用该规则的服务。

- 速率（0 到 100 之间的整数）– 容器耗尽后，要检测的匹配请求的百分比。在控制台中配置采样规则时，请选择 0 到 100 之间的百分比。使用 JSON 文档在客户端 SDK 中配置采样时，请提供介于 0 和 1 之间的一个百分比。
- 服务名称（字符串）- 检测过的服务在跟踪地图中显示的名称。
  - X-Ray SDK - 您在记录器上配置的服务名称。
  - Amazon API Gateway - *api-name*/*stage*。
- 服务类型（字符串）- 在跟踪地图中显示的服务类型。对于 X-Ray SDK，请通过应用合适的插件来设置服务类型：
  - `AWS::ElasticBeanstalk::Environment`— AWS Elastic Beanstalk 环境（插件）。
  - `AWS::EC2::Instance`— 亚马逊 EC2 实例（插件）。
  - `AWS::ECS::Container` — Amazon ECS 容器（插件）。
  - `AWS::APIGateway::Stage` - Amazon API Gateway 阶段。
  - `AWS::AppSync::GraphQLAPI` — 一个 AWS AppSync API 请求。
- 主机（字符串）— HTTP 主机标头中的主机名。
- HTTP 方法 - 字符串 HTTP 请求的方法。
- URL 路径（字符串）— 请求的 URL 路径。
  - X-Ray SDK – HTTP 请求 URL 的路径部分。
- 资源 ARN（字符串）-运行服务的 AWS 资源的 ARN。
  - X-Ray 开发工具包 — 不支持。SDK 只能使用资源 ARN 设置为 * 的规则。
  - Amazon API Gateway - 阶段 ARN。
- （可选）属性（键和值）- 在做出采样决定时已知的片段属性。
  - X-Ray 开发工具包 — 不支持。该 SDK 将忽略指定属性的规则。
  - AmazonAPI Gateway - 来自原始 HTTP 请求的标头。

采样规则示例

Example - 没有容器和低比率的默认规则

您可以修改默认规则的容器和比率。默认规则应用于与任何其他规则都不匹配的请求。

- 容器：**0**
- 速率：**5**（使用是使用的 JSON 文档配置的 **0.05**）

Example - 调试规则以跟踪对有问题的路由的所有请求

一个临时应用的用于调试的高优先级规则。

- 规则名称：**DEBUG - history updates**
- 优先级：**1**
- 容器：**1**
- 速率：**100**（使用是使用的 JSON 文档配置的 **1**）
- 服务名称：**Scorekeep**
- 服务类型：*****
- 主机：*****
- HTTP 方法：**PUT**
- URL 路径：**/history/***
- 资源 ARN：*****

Example — 更高的最低费率 POSTs

- 规则名称：**POST minimum**
- 优先级：**100**
- 容器：**10**
- 速率：**10**（使用是使用的 JSON 文档配置的 **.1**）
- 服务名称：*****
- 服务类型：*****
- 主机：*****
- HTTP 方法：**POST**
- URL 路径：*****
- 资源 ARN：*****

将服务配置为使用采样规则

X-Ray SDK 需要额外配置来使用在控制台中配置的采样规则。有关更多信息，请参阅采用您的语言的配置主题中有关配置采样策略的详细信息：

- Java：采样规则

- Go：采样规则
- Node.js：采样规则
- Python：采样规则
- Ruby：采样规则
- .NET：采样规则

对于 API 网关，请参阅Amazon API Gateway 主动追踪支持 AWS X-Ray。

查看采样结果

X-Ray 控制台采样页显示了有关您的服务如何使用每个采样规则的详细信息。

趋势列显示了在前几分钟如何使用了规则。每个列显示了 10 秒时段的统计数据。

采样统计数据

- 匹配的总规则数：与此规则匹配的请求数。此数字不包含可能与此规则匹配但先与优先级更高的规则匹配的请求。
- 总采样数：已记录的请求数。
- 以固定比率采样：通过应用规则的固定比率采样的请求数。
- 在容器限制下采样：使用由 X-Ray 分配的配额采样的请求数。
- 已从容器借用:通过从容器借用来采样的请求数。当某个服务首次将请求与规则匹配时，X-Ray 尚未向它分配配额。但是，如果容器至少为 1，该服务会每秒借用一个跟踪，直到 X-Ray 分配一个配额。

有关采样统计数据以及服务采样规则的方式的更多信息，请参阅通过 X-Ray API 使用采样规则。

后续步骤

您可以使用 X-Ray API 管理采样规则。利用 API，您可以按计划以编程方式创建和更新规则，也可以作为对警报或通知的响应执行此操作。有关说明和其他规则示例，请参阅利用 AWS X-Ray API 配置采样、组和加密设置。

X-Ray SDK AWS 服务 还使用 X-Ray API 来读取采样规则、报告采样结果和获取采样目标。服务必须跟踪它们应用每个规则的频率，根据优先级评估规则，并在某个请求与 X-Ray 尚未针对其向服务分配配额的规则匹配时从容器中借用。有关服务如何使用 API 进行采样的更多详细信息，请参阅通过 X-Ray API 使用采样规则。

当 X-Ray SDK 调用采样时 APIs，它会使用 X-Ray 守护程序作为代理。如果您已使用 TCP 端口 2000，则可以配置进程守护程序以在其他端口上运行代理。有关详细信息，请参阅配置 AWS X-Ray 守护程序。

控制台深层链接

可以使用路由和查询深层链接到特定跟踪，或者筛选跟踪和跟踪地图的视图。

控制台页面

- 欢迎页面 — xray/home#/welcome
- 入门 — — xray/home#/getting 入门
- 追踪地图 — xray/home#/service- map
- 痕迹 — xray/home#/traces

跟踪

您可以针对每个跟踪的时间线、原始和映射视图生成链接。

跟踪时间线 - `xray/home#/traces/`*`trace-id`*

原始跟踪数据 - `xray/home#/traces/`*`trace-id`*`/raw`

Example - 原始跟踪数据

```
https://console.aws.amazon.com/xray/home#/traces/1-57f5498f-d91047849216d0f2ea3b6442/
raw
```

筛选条件表达式

链接到筛选的跟踪列表。

筛选的跟踪视图 - `xray/home#/traces?filter=`*`filter-expression`*

Example - 筛选表达式

```
https://console.aws.amazon.com/xray/home#/traces?filter=service("api.amazon.com")
 { fault = true OR responsetime > 2.5 } AND annotation.foo = "bar"
```

Example - 筛选条件表达式（URL 编码）

```
https://console.aws.amazon.com/xray/home#/traces?filter=service(%22api.amazon.com
%22)%20%7B%20fault%20%3D%20true%20OR%20responsetime%20%3E%202.5%20%7D%20AND
%20annotation.foo%20%3D%20%22bar%22
```

有关筛选条件表达式的更多信息，请参阅使用筛选条件表达式。

时间范围

以 ISO86 01 格式指定时间长度或开始和结束时间。时间范围采用 UTC，最长可达 6 小时。

时间长度 - `xray/home#/`*`page`*`?timeRange=`*`range-in-minutes`*

Example - 最近 1 小时的跟踪地图

```
https://console.aws.amazon.com/xray/home#/service-map?timeRange=PT1H
```

开始和结束时间 - `xray/home#/`*`page`*`?timeRange=`*`start`*`~`*`end`*

Example - 时间范围精确到秒

```
https://console.aws.amazon.com/xray/home#/traces?
timeRange=2023-7-01T16:00:00~2023-7-01T22:00:00
```

Example - 时间范围精确到分钟

```
https://console.aws.amazon.com/xray/home#/traces?
timeRange=2023-7-01T16:00~2023-7-01T22:00
```

区域

指定一个 AWS 区域 以链接到该区域的页面。如果您未指定区域，则控制台会将您重定向到最近访问过的区域。

区域 - `xray/home`**`?region=`*`region`***`#/`*`page`*

Example - 美国西部（俄勒冈州）（us-west-2）的跟踪地图

```
https://console.aws.amazon.com/xray/home?region=us-west-2#/service-map
```

当您将带有其他查询参数的 Region 包括在内时，Region 查询位于哈希值之前， X-Ray-specific查询位于页面名称之后。

Example - 美国西部（俄勒冈州）（us-west-2）最近 1 小时的跟踪地图

```
https://console.aws.amazon.com/xray/home?region=us-west-2#/service-map?timeRange=PT1H
```

组合

Example - 使用持续时间筛选条件的最近跟踪

```
https://console.aws.amazon.com/xray/home#/traces?timeRange=PT15M&filter=duration%20%3E
%3D%205%20AND%20duration%20%3C%3D%208
```

输出

- 页面 - 跟踪
- 时间范围 - 过去 15 分钟
- 筛选条件 - duration >= 5 AND duration <= 8

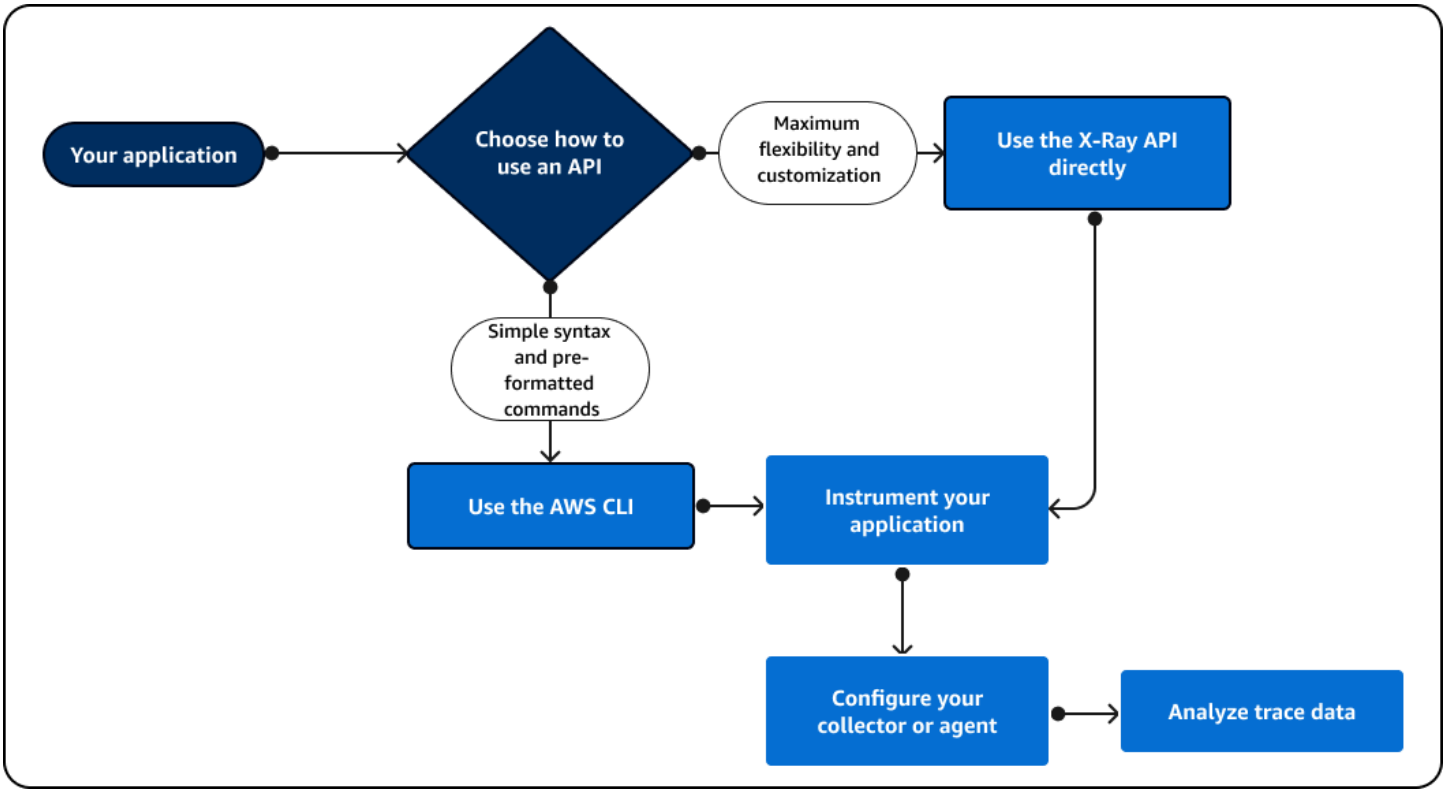
使用 X-Ray API

如果 X-Ray SDK 不支持你的编程语言，你可以 APIs 直接使用 X-Ray 或 AWS Command Line Interface (AWS CLI) 来调用 X-Ray API 命令。使用以下指南来选择与 API 的交互方式：

- 使用 AWS CLI 预先格式化的命令或请求中的选项来简化语法。
- 直接使用 X-Ray API，以大幅提高灵活性，并根据您向 X-Ray 提出的请求进行自定义。

如果您直接使用 X-Ray API 而不是 AWS CLI，则必须以正确的数据格式对请求进行参数化，可能还必须配置身份验证和错误处理。

下图显示了相关指南，可帮助您选择与 X-Ray API 的交互方式：



使用 X-Ray API 将跟踪数据直接发送到 X-Ray。X-Ray API 支持 X-Ray SDK 中提供的所有功能，包括以下常见操作：

- PutTraceSegments— 将分段文档上传到 X-Ray。
- BatchGetTraces— 检索跟踪列表中的跟踪 IDs列表。检索到的每个跟踪是一组来自单个请求的分段文档。
- GetTraceSummaries— 检索轨迹 IDs 并对其进行注释。您可以指定 `FilterExpression` 来检索跟踪摘要的子集。
- GetTraceGraph— 检索特定跟踪 ID 的服务图表。
- GetServiceGraph— 检索 JSON 格式化文档，描述处理传入请求和调用下游请求的服务。

您还可以在应用程序代码中使用 AWS Command Line Interface (AWS CLI) 以编程方式与 X-Ray 进行交互。 AWS CLI 支持 X-Ray SDK 中可用的所有功能，包括其他功能 AWS 服务。以下函数是前面列出的 API 操作的版本，格式更简单：

- put-trace-segments— 将分段文档上传到 X-Ray。
- batch-get-traces— 检索跟踪列表中的跟踪 IDs列表。检索到的每个跟踪是一组来自单个请求的分段文档。

- get-trace-summaries— 检索轨迹 IDs 并对其进行注释。您可以指定 `FilterExpression` 来检索跟踪摘要的子集。
- get-trace-graph— 检索特定跟踪 ID 的服务图表。
- get-service-graph— 检索JSON格式化文档，该文档描述了处理传入请求和调用下游请求的服务。

要开始使用，您必须AWS CLI为自己的操作系统安装。 AWS 支持 Linux, macOS 以及 Windows 操作系统。有关 X-Ray 命令列表的更多信息，请参阅针对 X-Ray 的AWS CLI 命令参考指南。

主题

- 将 AWS X-Ray API 与 CLI 配合 AWS 使用
- 将跟踪数据发送到 AWS X-Ray
- 从中获取数据 AWS X-Ray
- 利用 AWS X-Ray API 配置采样、组和加密设置
- 通过 X-Ray API 使用采样规则
- AWS X-Ray 分段文档

将 AWS X-Ray API 与 CLI 配合 AWS 使用

AWS CLI 允许您直接访问 X-Ray 服务 APIs ，并使用与 X-Ray 控制台相同的服务来检索服务图表和原始跟踪数据。示例应用程序包括显示如何在 AWS CLI 中 APIs 使用这些脚本。

先决条件

本教程使用 Scorekeep 示例应用程序并包括了用于生成跟踪数据和服务地图的脚本。按照入门教程中的说明启动应用程序。

本教程使用 AWS CLI 来演示 X-Ray API 的基本用法。适用于 Windows、 AWS Linux 和 OS-X 的 CLI APIs 为所有公众提供了命令行访问权限。 AWS 服务

> **Note**
>
> 您必须验证您的配置 AWS CLI 是否与创建 Scorekeeep 示例应用程序所在的区域相同。

其中包括测试示例应用程序的脚本，该脚本使用 cURL 发送流量到 API 和 `jq` 来解析输出。您可以从 jqstedolan.github.io 下载 可执行文件，从 `curl` 下载 https://curl.haxx.se/download.html 可执行文件。大部分 Linux 和 OS X 安装包含 cURL。

生成跟踪数据

Web 应用程序在游戏进行中每几秒继续生成对 API 的流量，但仅生成一种类型的请求。在您测试 API 时，使用 `test-api.sh` 脚本运行端到端方案并生成更多样的跟踪数据。

使用 **`test-api.sh`** 脚本

1. 打开 Elastic Beanstalk 控制台。
2. 导航到您的环境的管理控制台。
3. 从页面标题复制环境 URL。
4. 打开 `bin/test-api.sh` 并使用您环境的 URL 替换 API 的值。

```
#!/bin/bash
API=scorekeep.9hbtbm23t2.us-west-2.elasticbeanstalk.com/api
```

5. 运行脚本以生成对 API 的流量。

```
~/debugger-tutorial$ ./bin/test-api.sh
Creating users,
session,
game,
configuring game,
playing game,
ending game,
game complete.
{"id":"MTBP8BAS","session":"HUF6IT64","name":"tic-tac-toe-test","users":
["QFF3HBGM","KL6JR98D"],"rules":"102","startTime":1476314241,"endTime":1476314245,"states":
["JQVLEOM2","D67QLPIC","VF9BM9NC","OEAA6GK9","2A705O73","1U2LFTLJ","HUKIDD70","BAN1C8FI","G
["BS8F8LQ","4MTTSPKP","463OETES","SVEBCL3N","N7CQ1GHP","O84ONEPD","EG4BPROQ","V4BLIDJ3","9R
```

使用 X-Ray API

AWS CLI 为 X-Ray 提供的所有 API 操作提供命令，包括`GetServiceGraph`和`GetTraceSummaries`。有关所有支持的操作以及这些操作所使用数据类型的更多信息，请参阅 AWS X-Ray API 参考。

Example bin/service-graph.sh

```
EPOCH=$(date +%s)
aws xray get-service-graph --start-time $(($EPOCH-600)) --end-time $EPOCH
```

该脚本检索过去 10 分钟的服务图。

```
~/eb-java-scorekeep$ ./bin/service-graph.sh | less
{
    "StartTime": 1479068648.0,
    "Services": [
        {
            "StartTime": 1479068648.0,
            "ReferenceId": 0,
            "State": "unknown",
            "EndTime": 1479068651.0,
            "Type": "client",
            "Edges": [
                {
                    "StartTime": 1479068648.0,
                    "ReferenceId": 1,
                    "SummaryStatistics": {
                        "ErrorStatistics": {
                            "ThrottleCount": 0,
                            "TotalCount": 0,
                            "OtherCount": 0
                        },
                        "FaultStatistics": {
                            "TotalCount": 0,
                            "OtherCount": 0
                        },
                        "TotalCount": 2,
                        "OkCount": 2,
                        "TotalResponseTime": 0.054000139236450195
                    },
                    "EndTime": 1479068651.0,
                    "Aliases": []
                }
            ]
        },
        {
            "StartTime": 1479068648.0,
            "Names": [
```

```
                "scorekeep.elasticbeanstalk.com"
            ],
            "ReferenceId": 1,
            "State": "active",
            "EndTime": 1479068651.0,
            "Root": true,
            "Name": "scorekeep.elasticbeanstalk.com",
...
```

Example bin/trace-urls.sh

```
EPOCH=$(date +%s)
aws xray get-trace-summaries --start-time $(($EPOCH-120)) --end-time $(($EPOCH-60)) --
query 'TraceSummaries[*].Http.HttpURL'
```

该脚本会 URLs 检索一到两分钟前生成的跟踪。

```
~/eb-java-scorekeep$ ./bin/trace-urls.sh
[
    "http://scorekeep.elasticbeanstalk.com/api/game/6Q0UE1DG/5FGLM9U3/
endtime/1479069438",
    "http://scorekeep.elasticbeanstalk.com/api/session/KH4341QH",
    "http://scorekeep.elasticbeanstalk.com/api/game/GLQBJ3K5/153AHDIA",
    "http://scorekeep.elasticbeanstalk.com/api/game/VPDL672J/G2V41HM6/
endtime/1479069466"
]
```

Example bin/full-traces.sh

```
EPOCH=$(date +%s)
TRACEIDS=$(aws xray get-trace-summaries --start-time $(($EPOCH-120)) --end-time
 $(($EPOCH-60)) --query 'TraceSummaries[*].Id' --output text)
aws xray batch-get-traces --trace-ids $TRACEIDS --query 'Traces[*]'
```

该脚本检索在过去 1 到 2 分钟之间生成的完整跟踪。

```
~/eb-java-scorekeep$ ./bin/full-traces.sh | less
[
    {
        "Segments": [
            {
```

```
                "Id": "3f212bc237bafd5d",
                "Document": "{\"id\":\"3f212bc237bafd5d\",\"name\":\"DynamoDB\",
\"trace_id\":\"1-5828d9f2-a90669393f4343211bc1cf75\",\"start_time\":1.479072242459E9,
\"end_time\":1.479072242477E9,\"parent_id\":\"72a08dcf87991ca9\",\"http\":
{\"response\":{\"content_length\":60,\"status\":200}},\"inferred\":true,\"aws\":
{\"consistent_read\":false,\"table_name\":\"scorekeep-session-xray\",\"operation\":
\"GetItem\",\"request_id\":\"QAKE0S8DD0LJM245KAOPMA746BVV4KQNSO5AEMVJF66Q9ASUAAJG\",
\"resource_names\":[\"scorekeep-session-xray\"]},\"origin\":\"AWS::DynamoDB::Table\"}"
            },
            {
                "Id": "309e355f1148347f",
                "Document": "{\"id\":\"309e355f1148347f\",\"name\":\"DynamoDB\",
\"trace_id\":\"1-5828d9f2-a90669393f4343211bc1cf75\",\"start_time\":1.479072242477E9,
\"end_time\":1.479072242494E9,\"parent_id\":\"37f14ef837f00022\",\"http\":
{\"response\":{\"content_length\":606,\"status\":200}},\"inferred\":true,\"aws\":
{\"table_name\":\"scorekeep-game-xray\",\"operation\":\"UpdateItem\",\"request_id
\":\"388GEROC4PCA6D59ED3CTI5EEJVV4KQNSO5AEMVJF66Q9ASUAAJG\",\"resource_names\":
[\"scorekeep-game-xray\"]},\"origin\":\"AWS::DynamoDB::Table\"}"
            }
        ],
        "Id": "1-5828d9f2-a90669393f4343211bc1cf75",
        "Duration": 0.05099987983703613
    }
...
```


清理

终止您的 Elastic Beanstalk 环境以关闭 EC2 亚马逊实例、DynamoDB 表和其他资源。

终止 Elastic Beanstalk 环境

1. 打开 Elastic Beanstalk 控制台。
2. 导航到您的环境的管理控制台。
3. 选择操作。
4. 选择终止环境。
5. 选择终止。

在 30 之后，跟踪数据自动从 X-Ray 中删除。

将跟踪数据发送到 AWS X-Ray

您可以分段文档的形式将跟踪数据发送到 X-Ray。分段文档是 JSON 格式的字符串，其中包含有关您的应用程序在请求服务中所做工作的信息。您的应用程序可以将它自身所做工作的数据记录在分段中，将使用下游服务和资源的工作的数据记录在子分段中。

分段记录有关您的应用程序所做工作的信息。区段至少记录在一项任务上花费的时间、一个名称和两个 IDs。当请求在多个服务之间传输时，跟踪 ID 可对请求进行追踪。分段 ID 跟踪单个服务为请求所做的工作。

Example 最小完成分段

```
{
  "name" : "Scorekeep",
  "id" : "70de5b6f19ff9a0a",
  "start_time" : 1.478293361271E9,
  "trace_id" : "1-581cf771-a006649127e371903a2de979",
  "end_time" : 1.478293361449E9
}
```

当收到请求时，您可以发送正在运行的分段作为占位符，直到该请求完成。

Example 正在进行分段

```
{
  "name" : "Scorekeep",
  "id" : "70de5b6f19ff9a0b",
  "start_time" : 1.478293361271E9,
  "trace_id" : "1-581cf771-a006649127e371903a2de979",
  “in_progress”: true
}
```

您可以使用 PutTraceSegments 或通过 X-Ray 进程守护程序直接将分段发送给 X-Ray。

大多数应用程序使用 AWS SDK 调用其他服务或访问资源。在子分段中记录有关下游调用的信息。X-Ray 使用子分段来确定未发送分段的下游服务，并在服务图上为其创建条目。

子分段可以嵌入到完整分段文档或者单独发送。对于长时间运行的请求，单独发送子分段以异步跟踪下游调用，或者避免超过最大分段文档大小 (64KB)。

Example 子分段

子分段具有 type 的 subsegment 以及标识父分段的 parent_id。

```
{
  "name" : "www2.example.com",
  "id" : "70de5b6f19ff9a0c",
  "start_time" : 1.478293361271E9,
  "trace_id" : "1-581cf771-a006649127e371903a2de979"
  “end_time” : 1.478293361449E9,
  “type” : “subsegment”,
  “parent_id” : “70de5b6f19ff9a0b”
}
```

有关可包含在分段和子分段中的字段和值的更多信息，请参阅AWS X-Ray 分段文档。

Sections

- 正在生成跟踪 IDs
- 使用 PutTraceSegments
- 将分段文档发送到 X-Ray 进程守护程序

正在生成跟踪 IDs

要将数据发送到 X-Ray，必须为每个请求生成一个唯一的跟踪 ID。

X-Ray 跟踪 ID 格式

X-Ray trace_id 由以连字符分隔的三组数字组成。例如，1-58406520-a006649127e371903a2de979。这包括：

- 版本号，即 1。
- 原始请求的时间，采用 Unix 纪元时间，为 8 个十六进制数字。

  例如，2016 年 12 月 1 日上午 10:00（太平洋标准时间）的纪元时间为 1480615200 秒，或者是十六进制数字 58406520。
- 跟踪的 96 位全局唯一标识符，使用 24 个十六进制数字。

> **Note**
>
> X-Ray 现在支持使用创建的跟踪 IDs OpenTelemetry 以及任何其他符合 W3C 跟踪上下文规范的框架。发送到 X-Ray 时，W3C 跟踪 ID 必须采用 X-Ray 跟踪 ID 的格式。例如，W3C 跟踪 ID `4efaaf4d1e8720b39541901950019ee5` 在发送到 X-Ray 时，应与 `1-4efaaf4d-1e8720b39541901950019ee5` 的格式相同。X-Ray 跟踪 IDs 包括以 Unix 纪元时间为单位的原始请求时间戳，但是当以 IDs X-Ray 格式发送 W3C 跟踪时，这不是必需的。

您可以编写脚本来生成 X-Ray 跟踪 IDs 以供测试。以下是两个示例。

Python

```
import time
import os
import binascii

START_TIME = time.time()
HEX=hex(int(START_TIME))[2:]
TRACE_ID="1-{}-{}".format(HEX, binascii.hexlify(os.urandom(12)).decode('utf-8'))
```

Bash

```
START_TIME=$(date +%s)
HEX_TIME=$(printf '%x\n' $START_TIME)
GUID=$(dd if=/dev/random bs=12 count=1 2>/dev/null | od -An -tx1 | tr -d ' \t\n')
TRACE_ID="1-$HEX_TIME-$GUID"
```

有关创建跟踪 IDs 并将区段发送到 X-Ray 守护程序的脚本，请参阅 Scorekeeep 示例应用程序。

- Python – `xray_start.py`
- Bash - `xray_start.sh`

使用 PutTraceSegments

您可以使用 `PutTraceSegments` API 上传分段文档。该 API 只有一个参数 `TraceSegmentDocuments`，该参数采用 JSON 分段文档列表。

通过 AWS CLI，使用 `aws xray put-trace-segments` 命令将分段文档直接发送给 X-Ray。

```
$ DOC='{"trace_id": "1-5960082b-ab52431b496add878434aa25", "id": "6226467e3f845502",
 "start_time": 1498082657.37518, "end_time": 1498082695.4042, "name":
 "test.elasticbeanstalk.com"}'
$ aws xray put-trace-segments --trace-segment-documents "$DOC"
{
    "UnprocessedTraceSegments": []
}
```

> **Note**
>
> Windows 命令处理器和 Windows PowerShell 对 JSON 字符串中的引号和转义有不同的要求。有关详细信息，请参阅 用户指南中的为字符串加引号 AWS CLI 。

输出列出任何处理失败的分段。例如，如果跟踪 ID 中的日期是很久以前，您会看到一个如下所示的错误。

```
{
    "UnprocessedTraceSegments": [
        {
            "ErrorCode": "InvalidTraceId",
            "Message": "Invalid segment. ErrorCode: InvalidTraceId",
            "Id": "6226467e3f845502"
        }
    ]
}
```

您可以同时传递多个分段文档，中间用空格分隔。

```
$ aws xray put-trace-segments --trace-segment-documents "$DOC1" "$DOC2"
```

将分段文档发送到 X-Ray 进程守护程序

您可以不将分段文档发送到 X-Ray，而是将分段和子分段发送到 X-Ray 进程守护程序，进程守护程序将缓存它们，然后分批上传到 X-Ray API。X-Ray SDK 将分段文档发送到进程守护程序以避免直接调用 AWS 。

> **Note**
>
> 请参阅 在本地运行 X-Ray 进程守护程序 获得有关运行进程守护程序的说明。

通过 UDP 端口 2000 发送 JSON 格式分段，在前面加上进程守护程序标头 `{"format": "json", "version": 1}\n`

```
{"format": "json", "version": 1}\n{"trace_id": "1-5759e988-bd862e3fe1be46a994272793",
 "id": "defdfd9912dc5a56", "start_time": 1461096053.37518, "end_time": 1461096053.4042,
 "name": "test.elasticbeanstalk.com"}
```

在 Linux 上，您可以从 Bash 终端将分段文档发送给进程守护程序。将标头和分段文档保存到一个文本文件中，然后使用 `/dev/udp` 以管道形式传送到 `cat`。

```
$ cat segment.txt > /dev/udp/127.0.0.1/2000
```

Example segment.txt

```
{"format": "json", "version": 1}
{"trace_id": "1-594aed87-ad72e26896b3f9d3a27054bb", "id": "6226467e3f845502",
 "start_time": 1498082657.37518, "end_time": 1498082695.4042, "name":
 "test.elasticbeanstalk.com"}
```

检查进程守护程序日志，验证它是否已将分段发送到 X-Ray。

```
2017-07-07T01:57:24Z [Debug] processor: sending partial batch
2017-07-07T01:57:24Z [Debug] processor: segment batch size: 1. capacity: 50
2017-07-07T01:57:24Z [Info] Successfully sent batch of 1 segments (0.020 seconds)
```

从中获取数据 AWS X-Ray

AWS X-Ray 处理您发送给它的跟踪数据，以生成 JSON 格式的完整跟踪、跟踪摘要和服务图。您可以使用 AWS CLI 直接从 API 检索生成的数据。

Sections

- 检索和细化根本原因分析

检索服务图

您可以使用 GetServiceGraph API 来检索 JSON 服务图。该 API 需要开始时间和结束时间，您可以使用 date 命令从 Linux 终端计算这些时间。

```
$ date +%s
1499394617
```

date +%s 显示日期（秒数）。使用该数字作为结束时间，并从中减去一个时间可得到开始时间。

Example 用于检索最后 10 分钟的服务图的脚本

```
EPOCH=$(date +%s)
aws xray get-service-graph --start-time $(($EPOCH-600)) --end-time $EPOCH
```

以下示例显示了一个包含 4 个节点的服务图，包括一个客户端节点、一个 EC2 实例、一个 DynamoDB 表和一个 Amazon SNS 主题。

Example GetServiceGraph 输出

```
{
    "Services": [
        {
            "ReferenceId": 0,
            "Name": "xray-sample.elasticbeanstalk.com",
            "Names": [
                "xray-sample.elasticbeanstalk.com"
            ],
            "Type": "client",
            "State": "unknown",
            "StartTime": 1528317567.0,
            "EndTime": 1528317589.0,
            "Edges": [
                {
                    "ReferenceId": 2,
                    "StartTime": 1528317567.0,
                    "EndTime": 1528317589.0,
                    "SummaryStatistics": {
                        "OkCount": 3,
                        "ErrorStatistics": {
```

```
                        "ThrottleCount": 0,
                        "OtherCount": 1,
                        "TotalCount": 1
                    },
                    "FaultStatistics": {
                        "OtherCount": 0,
                        "TotalCount": 0
                    },
                    "TotalCount": 4,
                    "TotalResponseTime": 0.273
                },
                "ResponseTimeHistogram": [
                    {
                        "Value": 0.005,
                        "Count": 1
                    },
                    {
                        "Value": 0.015,
                        "Count": 1
                    },
                    {
                        "Value": 0.157,
                        "Count": 1
                    },
                    {
                        "Value": 0.096,
                        "Count": 1
                    }
                ],
                "Aliases": []
            }
        ]
    },
    {
        "ReferenceId": 1,
        "Name": "awseb-e-dixzws4s9p-stack-StartupSignupsTable-4IMSMHAYX2BA",
        "Names": [
            "awseb-e-dixzws4s9p-stack-StartupSignupsTable-4IMSMHAYX2BA"
        ],
        "Type": "AWS::DynamoDB::Table",
        "State": "unknown",
        "StartTime": 1528317583.0,
        "EndTime": 1528317589.0,
        "Edges": [],
```

```
            "SummaryStatistics": {
                "OkCount": 2,
                "ErrorStatistics": {
                    "ThrottleCount": 0,
                    "OtherCount": 0,
                    "TotalCount": 0
                },
                "FaultStatistics": {
                    "OtherCount": 0,
                    "TotalCount": 0
                },
                "TotalCount": 2,
                "TotalResponseTime": 0.12
            },
            "DurationHistogram": [
                {
                    "Value": 0.076,
                    "Count": 1
                },
                {
                    "Value": 0.044,
                    "Count": 1
                }
            ],
            "ResponseTimeHistogram": [
                {
                    "Value": 0.076,
                    "Count": 1
                },
                {
                    "Value": 0.044,
                    "Count": 1
                }
            ]
        },
        {
            "ReferenceId": 2,
            "Name": "xray-sample.elasticbeanstalk.com",
            "Names": [
                "xray-sample.elasticbeanstalk.com"
            ],
            "Root": true,
            "Type": "AWS::EC2::Instance",
            "State": "active",
```

```
            "StartTime": 1528317567.0,
            "EndTime": 1528317589.0,
            "Edges": [
                {
                    "ReferenceId": 1,
                    "StartTime": 1528317567.0,
                    "EndTime": 1528317589.0,
                    "SummaryStatistics": {
                        "OkCount": 2,
                        "ErrorStatistics": {
                            "ThrottleCount": 0,
                            "OtherCount": 0,
                            "TotalCount": 0
                        },
                        "FaultStatistics": {
                            "OtherCount": 0,
                            "TotalCount": 0
                        },
                        "TotalCount": 2,
                        "TotalResponseTime": 0.12
                    },
                    "ResponseTimeHistogram": [
                        {
                            "Value": 0.076,
                            "Count": 1
                        },
                        {
                            "Value": 0.044,
                            "Count": 1
                        }
                    ],
                    "Aliases": []
                },
                {
                    "ReferenceId": 3,
                    "StartTime": 1528317567.0,
                    "EndTime": 1528317589.0,
                    "SummaryStatistics": {
                        "OkCount": 2,
                        "ErrorStatistics": {
                            "ThrottleCount": 0,
                            "OtherCount": 0,
                            "TotalCount": 0
                        },
```

```
                    "FaultStatistics": {
                        "OtherCount": 0,
                        "TotalCount": 0
                    },
                    "TotalCount": 2,
                    "TotalResponseTime": 0.125
                },
                "ResponseTimeHistogram": [
                    {
                        "Value": 0.049,
                        "Count": 1
                    },
                    {
                        "Value": 0.076,
                        "Count": 1
                    }
                ],
                "Aliases": []
            }
        ],
        "SummaryStatistics": {
            "OkCount": 3,
            "ErrorStatistics": {
                "ThrottleCount": 0,
                "OtherCount": 1,
                "TotalCount": 1
            },
            "FaultStatistics": {
                "OtherCount": 0,
                "TotalCount": 0
            },
            "TotalCount": 4,
            "TotalResponseTime": 0.273
        },
        "DurationHistogram": [
            {
                "Value": 0.005,
                "Count": 1
            },
            {
                "Value": 0.015,
                "Count": 1
            },
            {
```

```
                    "Value": 0.157,
                    "Count": 1
                },
                {
                    "Value": 0.096,
                    "Count": 1
                }
            ],
            "ResponseTimeHistogram": [
                {
                    "Value": 0.005,
                    "Count": 1
                },
                {
                    "Value": 0.015,
                    "Count": 1
                },
                {
                    "Value": 0.157,
                    "Count": 1
                },
                {
                    "Value": 0.096,
                    "Count": 1
                }
            ]
        },
        {
            "ReferenceId": 3,
            "Name": "SNS",
            "Names": [
                "SNS"
            ],
            "Type": "AWS::SNS",
            "State": "unknown",
            "StartTime": 1528317583.0,
            "EndTime": 1528317589.0,
            "Edges": [],
            "SummaryStatistics": {
                "OkCount": 2,
                "ErrorStatistics": {
                    "ThrottleCount": 0,
                    "OtherCount": 0,
                    "TotalCount": 0
```

```
                },
                "FaultStatistics": {
                    "OtherCount": 0,
                    "TotalCount": 0
                },
                "TotalCount": 2,
                "TotalResponseTime": 0.125
            },
            "DurationHistogram": [
                {
                    "Value": 0.049,
                    "Count": 1
                },
                {
                    "Value": 0.076,
                    "Count": 1
                }
            ],
            "ResponseTimeHistogram": [
                {
                    "Value": 0.049,
                    "Count": 1
                },
                {
                    "Value": 0.076,
                    "Count": 1
                }
            ]
        }
    ]
}
```

按组检索服务图

要根据组的内容调用服务图，请包括 groupName 或 groupARN。以下示例显示的是对一个名为“Example1”的组进行服务图调用。

Example 用于按组 Example1 的名称检索服务图的脚本

```
aws xray get-service-graph --group-name "Example1"
```

检索跟踪

您可以使用 GetTraceSummaries API 获取跟踪摘要列表。跟踪摘要包含可用于识别要完整下载的跟踪的信息，包括注释、请求和响应信息，以及 IDs。

调用 `aws xray get-trace-summaries` 时可以使用两个 TimeRangeType 标志：

- TraceId— 默认GetTraceSummaries搜索使用 TraceID 时间并返回在计算[start_time, end_time)范围内开始的轨迹。此时间戳范围是根据内时间戳的编码计算得出的 TraceId，也可以手动定义。
- 事件时间 - 用于搜索随着时间发生的事件，X AWS -Ray 允许使用事件时间戳搜索跟踪。无论追踪何时开始，事件时间都会返回 [start_time, end_time) 范围内处于活动状态的跟踪。

使用 `aws xray get-trace-summaries` 命令获取跟踪摘要列表。以下命令使用默认 TraceId时间获取过去 1 到 2 分钟的跟踪摘要列表。

Example 用于获取跟踪摘要的脚本

```
EPOCH=$(date +%s)
aws xray get-trace-summaries --start-time $(($EPOCH-120)) --end-time $(($EPOCH-60))
```

Example GetTraceSummaries 输出

```
{
    "TraceSummaries": [
        {
            "HasError": false,
            "Http": {
                "HttpStatus": 200,
                "ClientIp": "205.255.255.183",
                "HttpURL": "http://scorekeep.elasticbeanstalk.com/api/session",
                "UserAgent": "Mozilla/5.0 (Windows NT 6.1; Win64; x64)
 AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.115 Safari/537.36",
                "HttpMethod": "POST"
            },
            "Users": [],
            "HasFault": false,
            "Annotations": {},
            "ResponseTime": 0.084,
            "Duration": 0.084,
            "Id": "1-59602606-a43a1ac52fc7ee0eea12a82c",
```

```
            "HasThrottle": false
        },
        {
            "HasError": false,
            "Http": {
                "HttpStatus": 200,
                "ClientIp": "205.255.255.183",
                "HttpURL": "http://scorekeep.elasticbeanstalk.com/api/user",
                "UserAgent": "Mozilla/5.0 (Windows NT 6.1; Win64; x64)
 AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.115 Safari/537.36",
                "HttpMethod": "POST"
            },
            "Users": [
                {
                    "UserName": "5M388M1E"
                }
            ],
            "HasFault": false,
            "Annotations": {
                "UserID": [
                    {
                        "AnnotationValue": {
                            "StringValue": "5M388M1E"
                        }
                    }
                ],
                "Name": [
                    {
                        "AnnotationValue": {
                            "StringValue": "Ola"
                        }
                    }
                ]
            },
            "ResponseTime": 3.232,
            "Duration": 3.232,
            "Id": "1-59602603-23fc5b688855d396af79b496",
            "HasThrottle": false
        }
    ],
    "ApproximateTime": 1499473304.0,
    "TracesProcessedCount": 2
}
```

使用来自输出的跟踪 ID 通过 BatchGetTraces API 来检索完整跟踪。

Example BatchGetTraces 命令

```
$ aws xray batch-get-traces --trace-ids 1-596025b4-7170afe49f7aa708b1dd4a6b
```

Example BatchGetTraces 输出


```
{
    "Traces": [
        {
            "Duration": 3.232,
            "Segments": [
                {
                    "Document": "{\"id\":\"1fb07842d944e714\",\"name\":
\"random-name\",\"start_time\":1.499473411677E9,\"end_time\":1.499473414572E9,
\"parent_id\":\"0c544c1b1bbff948\",\"http\":{\"response\":{\"status\":200}},
\"aws\":{\"request_id\":\"ac086670-6373-11e7-a174-f31b3397f190\"},\"trace_id\":
\"1-59602603-23fc5b688855d396af79b496\",\"origin\":\"AWS::Lambda\",\"resource_arn\":
\"arn:aws:lambda:us-west-2:123456789012:function:random-name\"}",
                    "Id": "1fb07842d944e714"
                },
                {
                    "Document": "{\"id\":\"194fcc8747581230\",\"name\":\"Scorekeep
\",\"start_time\":1.499473411562E9,\"end_time\":1.499473414794E9,\"http\":{\"request
\":{\"url\":\"http://scorekeep.elasticbeanstalk.com/api/user\",\"method\":\"POST\",
\"user_agent\":\"Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML,
 like Gecko) Chrome/59.0.3071.115 Safari/537.36\",\"client_ip\":\"205.251.233.183\"},
\"response\":{\"status\":200}},\"aws\":{\"elastic_beanstalk\":{\"version_label\":\"app-
abb9-170708_002045\",\"deployment_id\":406,\"environment_name\":\"scorekeep-dev\"},
\"ec2\":{\"availability_zone\":\"us-west-2c\",\"instance_id\":\"i-0cd9e448944061b4a
\"},\"xray\":{\"sdk_version\":\"1.1.2\",\"sdk\":\"X-Ray for Java\"}},\"service
\":{},\"trace_id\":\"1-59602603-23fc5b688855d396af79b496\",\"user\":\"5M388M1E
\",\"origin\":\"AWS::ElasticBeanstalk::Environment\",\"subsegments\":[{\"id\":
\"0c544c1b1bbff948\",\"name\":\"Lambda\",\"start_time\":1.499473411629E9,\"end_time
\":1.499473414572E9,\"http\":{\"response\":{\"status\":200,\"content_length\":14}},
\"aws\":{\"log_type\":\"None\",\"status_code\":200,\"function_name\":\"random-name
\",\"invocation_type\":\"RequestResponse\",\"operation\":\"Invoke\",\"request_id
\":\"ac086670-6373-11e7-a174-f31b3397f190\",\"resource_names\":[\"random-name\"]},
\"namespace\":\"aws\"},{\"id\":\"071684f2e555e571\",\"name\":\"## UserModel.saveUser
\",\"start_time\":1.499473414581E9,\"end_time\":1.499473414769E9,\"metadata\":{\"debug
\":{\"test\":\"Metadata string from UserModel.saveUser\"}},\"subsegments\":[{\"id\":
\"4cd3f10b76c624b4\",\"name\":\"DynamoDB\",\"start_time\":1.49947341469E9,\"end_time
```

\":1.499473414769E9,\"http\":{\"response\":{\"status\":200,\"content_length\":57}},
\"aws\":{\"table_name\":\"scorekeep-user\",\"operation\":\"UpdateItem\",\"request_id
\":\"MFQ8CGJ3JTDDVVVASUAAJGQ6NJ82F738BOB4KQNSO5AEMVJF66Q9\",\"resource_names\":
[\"scorekeep-user\"]},\"namespace\":\"aws\"}]}]}",
                    "Id": "194fcc8747581230"
                },
                {
                    "Document": "{\"id\":\"00f91aa01f4984fd\",\"name\":
\"random-name\",\"start_time\":1.49947341283E9,\"end_time\":1.49947341457E9,
\"parent_id\":\"1fb07842d944e714\",\"aws\":{\"function_arn\":\"arn:aws:lambda:us-
west-2:123456789012:function:random-name\",\"resource_names\":[\"random-name\"],
\"account_id\":\"123456789012\"},\"trace_id\":\"1-59602603-23fc5b688855d396af79b496\",
\"origin\":\"AWS::Lambda::Function\",\"subsegments\":[{\"id\":\"e6d2fe619f827804\",
\"name\":\"annotations\",\"start_time\":1.499473413012E9,\"end_time\":1.499473413069E9,
\"annotations\":{\"UserID\":\"5M388M1E\",\"Name\":\"Ola\"}},{\"id\":\"b29b548af4d54a0f
\",\"name\":\"SNS\",\"start_time\":1.499473413112E9,\"end_time\":1.499473414071E9,
\"http\":{\"response\":{\"status\":200}},\"aws\":{\"operation\":\"Publish\",
\"region\":\"us-west-2\",\"request_id\":\"a2137970-f6fc-5029-83e8-28aadeb99198\",
\"retries\":0,\"topic_arn\":\"arn:aws:sns:us-west-2:123456789012:awseb-e-
ruag3jyweb-stack-NotificationTopic-6B829NT9V5O9\"},\"namespace\":\"aws\"},{\"id\":
\"2279c0030c955e52\",\"name\":\"Initialization\",\"start_time\":1.499473412064E9,
\"end_time\":1.499473412819E9,\"aws\":{\"function_arn\":\"arn:aws:lambda:us-
west-2:123456789012:function:random-name\"}}]}",
                    "Id": "00f91aa01f4984fd"
                },
                {
                    "Document": "{\"id\":\"17ba309b32c7fbaf\",\"name\":
\"DynamoDB\",\"start_time\":1.49947341469E9,\"end_time\":1.499473414769E9,
\"parent_id\":\"4cd3f10b76c624b4\",\"inferred\":true,\"http\":{\"response
\":{\"status\":200,\"content_length\":57}},\"aws\":{\"table_name
\":\"scorekeep-user\",\"operation\":\"UpdateItem\",\"request_id\":
\"MFQ8CGJ3JTDDVVVASUAAJGQ6NJ82F738BOB4KQNSO5AEMVJF66Q9\",\"resource_names\":
[\"scorekeep-user\"]},\"trace_id\":\"1-59602603-23fc5b688855d396af79b496\",\"origin\":
\"AWS::DynamoDB::Table\"}",
                    "Id": "17ba309b32c7fbaf"
                },
                {
                    "Document": "{\"id\":\"1ee3c4a523f89ca5\",\"name\":\"SNS
\",\"start_time\":1.499473413112E9,\"end_time\":1.499473414071E9,\"parent_id\":
\"b29b548af4d54a0f\",\"inferred\":true,\"http\":{\"response\":{\"status\":200}},\"aws
\":{\"operation\":\"Publish\",\"region\":\"us-west-2\",\"request_id\":\"a2137970-
f6fc-5029-83e8-28aadeb99198\",\"retries\":0,\"topic_arn\":\"arn:aws:sns:us-
west-2:123456789012:awseb-e-ruag3jyweb-stack-NotificationTopic-6B829NT9V5O9\"},
\"trace_id\":\"1-59602603-23fc5b688855d396af79b496\",\"origin\":\"AWS::SNS\"}",

```
                    "Id": "1ee3c4a523f89ca5"
                }
            ],
            "Id": "1-59602603-23fc5b688855d396af79b496"
        }
    ],
    "UnprocessedTraceIds": []
}
```

完整跟踪为每个分段包括一个文档，该文档根据收到的具有相同跟踪 ID 的所有分段文档编译。这些文档并不等同于您的应用程序发送到 X-Ray 的原样数据。它们是 X-Ray 服务生成的、经过处理的文档。X-Ray 编译您的应用程序发送的分段文档，并删除不符合分段文档架构的数据，从而创建完整的跟踪文档。

X-Ray 还会为自身不发送分段的服务的下游调用创建推断分段。例如，当您通过检测过的客户端调用 DynamoDB 时，X-Ray SDK 会从其视角记录包含调用详细信息的子分段。但是，DynamoDB 不发送相应的分段。X-Ray 会使用子分段中的信息创建推断分段，以表示跟踪地图中的 DynamoDB 资源，并将其添加到跟踪文档中。

要从 API 获取多条跟踪，您需要一个跟踪列表 IDs，您可以通过AWS CLI 查询从`get-trace-summaries`的输出中提取该列表。将该列表重定向到 `batch-get-traces` 的输入可获取特定时间段的完整跟踪。

Example 用于获取一分钟时间段内完整跟踪的脚本

```
EPOCH=$(date +%s)
TRACEIDS=$(aws xray get-trace-summaries --start-time $(($EPOCH-120)) --end-time
 $(($EPOCH-60)) --query 'TraceSummaries[*].Id' --output text)
aws xray batch-get-traces --trace-ids $TRACEIDS --query 'Traces[*]'
```

检索和细化根本原因分析

使用 GetTraceSummaries API 生成跟踪摘要后，可以以 JSON 格式重复使用部分跟踪摘要，根据根本原因创建精细的筛选表达式。请参阅下面的示例了解细化步骤的演练。

Example GetTraceSummaries 输出示例-响应时间根本原因部分

```
{
  "Services": [
    {
      "Name": "GetWeatherData",
      "Names": ["GetWeatherData"],
```

```
      "AccountId": 123456789012,
      "Type": null,
      "Inferred": false,
      "EntityPath": [
        {
          "Name": "GetWeatherData",
          "Coverage": 1.0,
          'Remote": false
        },
        {
          "Name": "get_temperature",
          "Coverage": 0.8,
          "Remote": false
        }
      ]
    },
    {
      "Name": "GetTemperature",
      "Names": ["GetTemperature"],
      "AccountId": 123456789012,
      "Type": null,
      "Inferred": false,
      "EntityPath": [
        {
          "Name": "GetTemperature",
          "Coverage": 0.7,
          "Remote": false
        }
      ]
    }
  ]
}
```

通过编辑并忽略上述输出，此 JSON 可以成为匹配的根本原因实体的筛选条件。JSON 中显示的任何一个候选字段都必须准确，否则将不会返回跟踪。删除的字段将成为通配符值，且采用可与筛选表达式查询结构兼容的格式。

Example 重新格式化的响应时间根本原因

```
{
  "Services": [
    {
      "Name": "GetWeatherData",
```

```
      "EntityPath": [
        {
          "Name": "GetWeatherData"
        },
        {
          "Name": "get_temperature"
        }
      ]
    },
    {
      "Name": "GetTemperature",
      "EntityPath": [
        {
          "Name": "GetTemperature"
        }
      ]
    }
  ]
}
```

随后通过调用 `rootcause.json = #[{}]` 将此 JSON 用作筛选表达式的一部分。有关使用筛选表达式进行查询的更多详细信息，请参阅筛选表达式一章。

Example JSON 筛选条件示例

```
rootcause.json = #[{ "Services": [ { "Name": "GetWeatherData", "EntityPath": [{ "Name":
 "GetWeatherData" }, { "Name": "get_temperature" } ] }, { "Name": "GetTemperature",
 "EntityPath": [ { "Name": "GetTemperature" } ] } ] }]
```

利用 AWS X-Ray API 配置采样、组和加密设置

AWS X-Ray APIs 用于配置采样规则、组规则和加密设置。

Sections

- 加密设置
- 采样规则
- 组

加密设置

PutEncryptionConfig用于指定用于加密的 AWS Key Management Service (AWS KMS) 密钥。

> **Note**
>
> X-Ray 不支持非对称 KMS 密钥。

```
$ aws xray put-encryption-config --type KMS --key-id alias/aws/xray
{
    "EncryptionConfig": {
        "KeyId": "arn:aws:kms:us-east-2:123456789012:key/c234g4e8-39e9-4gb0-84e2-
b0ea215cbba5",
        "Status": "UPDATING",
        "Type": "KMS"
    }
}
```

对于密钥 ID，您可以使用别名（如示例中所示）、密钥 ID 或 Amazon 资源名称 (ARN)。

使用 GetEncryptionConfig 获取当前配置。X-Ray 应用设置后，状态将从 UPDATING 变为 ACTIVE。

```
$ aws xray get-encryption-config
{
    "EncryptionConfig": {
        "KeyId": "arn:aws:kms:us-east-2:123456789012:key/c234g4e8-39e9-4gb0-84e2-
b0ea215cbba5",
        "Status": "ACTIVE",
        "Type": "KMS"
    }
}
```

要停止使用 KMS 密钥并使用默认加密，请将加密类型设置为 NONE。

```
$ aws xray put-encryption-config --type NONE
{
    "EncryptionConfig": {
        "Status": "UPDATING",
        "Type": "NONE"
    }
}
```

采样规则

您可以使用 X-Ray API 管理账户中的采样规则。有关添加和管理标签的更多信息，请参阅标记 X-Ray 采样规则和组。

利用 GetSamplingRules 获取所有采样规则。

```
$ aws xray get-sampling-rules
{
    "SamplingRuleRecords": [
        {
            "SamplingRule": {
                "RuleName": "Default",
                "RuleARN": "arn:aws:xray:us-east-2:123456789012:sampling-rule/Default",
                "ResourceARN": "*",
                "Priority": 10000,
                "FixedRate": 0.05,
                "ReservoirSize": 1,
                "ServiceName": "*",
                "ServiceType": "*",
                "Host": "*",
                "HTTPMethod": "*",
                "URLPath": "*",
                "Version": 1,
                "Attributes": {}
            },
            "CreatedAt": 0.0,
            "ModifiedAt": 1529959993.0
        }
    ]
}
```

默认规则应用于所有与任何其他规则都不匹配的请求。这是优先级最低的规则，无法删除。但是，您可以使用 UpdateSamplingRule 更改速率和容器大小。

Example **UpdateSamplingRule** 的 API 输入 10000-default.json

```
{
    "SamplingRuleUpdate": {
        "RuleName": "Default",
        "FixedRate": 0.01,
        "ReservoirSize": 0
    }
```

```
}
```

以下示例使用前一个文件作为输入，将默认规则更改为没有容器的百分之一。标签是可选的。如果选择添加标签，则标签键是必填，标签值为选填。要从采样规则中移除现有标签，请使用 UntagResource

```
$ aws xray update-sampling-rule --cli-input-json file://1000-default.json --tags
 [{"Key": "key_name","Value": "value"},{"Key": "key_name","Value": "value"}]
{
    "SamplingRuleRecords": [
        {
            "SamplingRule": {
                "RuleName": "Default",
                "RuleARN": "arn:aws:xray:us-east-2:123456789012:sampling-rule/Default",
                "ResourceARN": "*",
                "Priority": 10000,
                "FixedRate": 0.01,
                "ReservoirSize": 0,
                "ServiceName": "*",
                "ServiceType": "*",
                "Host": "*",
                "HTTPMethod": "*",
                "URLPath": "*",
                "Version": 1,
                "Attributes": {}
            },
            "CreatedAt": 0.0,
            "ModifiedAt": 1529959993.0
        },
```

利用 CreateSamplingRule 创建更多采样规则。创建规则时，大多数规则字段都是必填字段。以下示例将创建两个规则。第一条规则为 Scorekeeep 示例应用程序设置了基本频率。它匹配 API 提供的所有不符合更高优先级规则的请求。

Example **UpdateSamplingRule** 的 API 输入 9000-base-scorekeep.json

```
{
    "SamplingRule": {
        "RuleName": "base-scorekeep",
        "ResourceARN": "*",
        "Priority": 9000,
        "FixedRate": 0.1,
        "ReservoirSize": 5,
```

```
        "ServiceName": "Scorekeep",
        "ServiceType": "*",
        "Host": "*",
        "HTTPMethod": "*",
        "URLPath": "*",
        "Version": 1
    }
}
```

第二条规则也应用于 Scorekeeep，但它的优先级更高，也更具体。此规则为轮询请求设置了非常低的采样率。这些是客户端每隔几秒钟发出的 GET 请求，用于检查游戏状态是否发生变化。

Example **UpdateSamplingRule** 的 API 输入 5000-polling-scorekeep.json

```
{
    "SamplingRule": {
        "RuleName": "polling-scorekeep",
        "ResourceARN": "*",
        "Priority": 5000,
        "FixedRate": 0.003,
        "ReservoirSize": 0,
        "ServiceName": "Scorekeep",
        "ServiceType": "*",
        "Host": "*",
        "HTTPMethod": "GET",
        "URLPath": "/api/state/*",
        "Version": 1
    }
}
```

标签是可选的。如果选择添加标签，则标签键是必填，标签值为选填。

```
$ aws xray create-sampling-rule --cli-input-json file://5000-polling-scorekeep.json --
tags [{"Key": "key_name","Value": "value"},{"Key": "key_name","Value": "value"}]
{
    "SamplingRuleRecord": {
        "SamplingRule": {
            "RuleName": "polling-scorekeep",
            "RuleARN": "arn:aws:xray:us-east-1:123456789012:sampling-rule/polling-
scorekeep",
            "ResourceARN": "*",
            "Priority": 5000,
            "FixedRate": 0.003,
```

```
            "ReservoirSize": 0,
            "ServiceName": "Scorekeep",
            "ServiceType": "*",
            "Host": "*",
            "HTTPMethod": "GET",
            "URLPath": "/api/state/*",
            "Version": 1,
            "Attributes": {}
        },
        "CreatedAt": 1530574399.0,
        "ModifiedAt": 1530574399.0
    }
}
$ aws xray create-sampling-rule --cli-input-json file://9000-base-scorekeep.json
{
    "SamplingRuleRecord": {
        "SamplingRule": {
            "RuleName": "base-scorekeep",
            "RuleARN": "arn:aws:xray:us-east-1:123456789012:sampling-rule/base-
scorekeep",
            "ResourceARN": "*",
            "Priority": 9000,
            "FixedRate": 0.1,
            "ReservoirSize": 5,
            "ServiceName": "Scorekeep",
            "ServiceType": "*",
            "Host": "*",
            "HTTPMethod": "*",
            "URLPath": "*",
            "Version": 1,
            "Attributes": {}
        },
        "CreatedAt": 1530574410.0,
        "ModifiedAt": 1530574410.0
    }
}
```

要删除采样规则，请使用 DeleteSamplingRule。

```
$ aws xray delete-sampling-rule --rule-name polling-scorekeep
{
    "SamplingRuleRecord": {
        "SamplingRule": {
```

```
            "RuleName": "polling-scorekeep",
            "RuleARN": "arn:aws:xray:us-east-1:123456789012:sampling-rule/polling-
scorekeep",
            "ResourceARN": "*",
            "Priority": 5000,
            "FixedRate": 0.003,
            "ReservoirSize": 0,
            "ServiceName": "Scorekeep",
            "ServiceType": "*",
            "Host": "*",
            "HTTPMethod": "GET",
            "URLPath": "/api/state/*",
            "Version": 1,
            "Attributes": {}
        },
        "CreatedAt": 1530574399.0,
        "ModifiedAt": 1530574399.0
    }
}
```

组

您可以使用 X-Ray API 管理您账户中的组。组是由筛选条件表达式定义的跟踪的集合。您可以使用群组来生成其他服务图表并提供 Amazon CloudWatch 指标。请参阅 从中获取数据 AWS X-Ray，以了解有关通过 X-Ray API 使用服务图和指标的更多详细信息。有关组的更多信息，请参阅 配置组。有关添加和管理标签的更多信息，请参阅标记 X-Ray 采样规则和组。

使用 `CreateGroup` 创建一个组。标签是可选的。如果选择添加标签，则标签键是必填，标签值为选填。

```
$ aws xray create-group --group-name "TestGroup" --filter-expression
 "service(\"example.com\") {fault}" --tags [{"Key": "key_name","Value": "value"},
{"Key": "key_name","Value": "value"}]
{
    "GroupName": "TestGroup",
    "GroupARN": "arn:aws:xray:us-east-2:123456789012:group/TestGroup/UniqueID",
    "FilterExpression": "service(\"example.com\") {fault OR error}"
}
```

获取所有包含 `GetGroups` 的现有组。

```
$ aws xray get-groups
```

```
{
    "Groups": [
        {
            "GroupName": "TestGroup",
            "GroupARN": "arn:aws:xray:us-east-2:123456789012:group/TestGroup/UniqueID",
            "FilterExpression": "service(\"example.com\") {fault OR error}"
        },
  {
            "GroupName": "TestGroup2",
            "GroupARN": "arn:aws:xray:us-east-2:123456789012:group/TestGroup2/
UniqueID",
            "FilterExpression": "responsetime > 2"
        }
    ],
 "NextToken": "tokenstring"
}
```

更新包含 UpdateGroup 的组。标签是可选的。如果选择添加标签，则标签键是必填，标签值为选填。要从群组中移除现有标签，请使用UntagResource。

```
$ aws xray update-group --group-name "TestGroup" --group-arn "arn:aws:xray:us-
east-2:123456789012:group/TestGroup/UniqueID" --filter-expression
 "service(\"example.com\") {fault OR error}" --tags [{"Key": "Stage","Value": "Prod"},
{"Key": "Department","Value": "QA"}]
{
    "GroupName": "TestGroup",
    "GroupARN": "arn:aws:xray:us-east-2:123456789012:group/TestGroup/UniqueID",
    "FilterExpression": "service(\"example.com\") {fault OR error}"
}
```

删除包含 DeleteGroup 的组。

```
$ aws xray delete-group --group-name "TestGroup" --group-arn "arn:aws:xray:us-
east-2:123456789012:group/TestGroup/UniqueID"
    {
    }
```

通过 X-Ray API 使用采样规则

AWS X-Ray SDK 使用 X-Ray API 来获取采样规则、报告采样结果和获取配额。您可以使用它们 APIs 来更好地了解采样规则的工作原理，或者使用 X-Ray SDK 不支持的语言实现采样。

首先利用 GetSamplingRules 获取所有采样规则。

```
$ aws xray get-sampling-rules
{
    "SamplingRuleRecords": [
        {
            "SamplingRule": {
                "RuleName": "Default",
                "RuleARN": "arn:aws:xray:us-east-1::sampling-rule/Default",
                "ResourceARN": "*",
                "Priority": 10000,
                "FixedRate": 0.01,
                "ReservoirSize": 0,
                "ServiceName": "*",
                "ServiceType": "*",
                "Host": "*",
                "HTTPMethod": "*",
                "URLPath": "*",
                "Version": 1,
                "Attributes": {}
            },
            "CreatedAt": 0.0,
            "ModifiedAt": 1530558121.0
        },
        {
            "SamplingRule": {
                "RuleName": "base-scorekeep",
                "RuleARN": "arn:aws:xray:us-east-1::sampling-rule/base-scorekeep",
                "ResourceARN": "*",
                "Priority": 9000,
                "FixedRate": 0.1,
                "ReservoirSize": 2,
                "ServiceName": "Scorekeep",
                "ServiceType": "*",
                "Host": "*",
                "HTTPMethod": "*",
                "URLPath": "*",
                "Version": 1,
                "Attributes": {}
            },
            "CreatedAt": 1530573954.0,
            "ModifiedAt": 1530920505.0
        },
        {
```

```
            "SamplingRule": {
                "RuleName": "polling-scorekeep",
                "RuleARN": "arn:aws:xray:us-east-1::sampling-rule/polling-scorekeep",
                "ResourceARN": "*",
                "Priority": 5000,
                "FixedRate": 0.003,
                "ReservoirSize": 0,
                "ServiceName": "Scorekeep",
                "ServiceType": "*",
                "Host": "*",
                "HTTPMethod": "GET",
                "URLPath": "/api/state/*",
                "Version": 1,
                "Attributes": {}
            },
            "CreatedAt": 1530918163.0,
            "ModifiedAt": 1530918163.0
        }
    ]
}
```

输出包括默认规则和自定义规则。如果还尚未创建采样规则，请参阅 采样规则。

根据传入请求按优先级升序评估规则。当规则匹配时，使用固定速率和容器大小来制定采样决定。记录采样请求并忽略出于跟踪目的的未采样请求。制定采样决定时停止评估规则。

规则容器大小由指在应用固定速率之前，每秒要记录的跟踪目标数量。容器累积应用于所有服务，因此无法直接使用。但是，如果它是非零值，您可以每秒从容器借用一个跟踪，直到 X-Ray 分配配额。在收到配额之前，请每秒记录第一个请求，然后将固定速率应用于其他请求。固定速率是介于 0 和 1.00 之间的小数 (100%)。

以下示例显示了对 GetSamplingTargets 的调用以及有关在过去 10 秒内所做的采样决定的详细信息。

```
$ aws xray get-sampling-targets --sampling-statistics-documents '[
    {
        "RuleName": "base-scorekeep",
        "ClientID": "ABCDEF1234567890ABCDEF10",
        "Timestamp": "2018-07-07T00:20:06",
        "RequestCount": 110,
        "SampledCount": 20,
```

```
        "BorrowCount": 10
    },
    {
        "RuleName": "polling-scorekeep",
        "ClientID": "ABCDEF1234567890ABCDEF10",
        "Timestamp": "2018-07-07T00:20:06",
        "RequestCount": 10500,
        "SampledCount": 31,
        "BorrowCount": 0
    }
]'
{
    "SamplingTargetDocuments": [
        {
            "RuleName": "base-scorekeep",
            "FixedRate": 0.1,
            "ReservoirQuota": 2,
            "ReservoirQuotaTTL": 1530923107.0,
            "Interval": 10
        },
        {
            "RuleName": "polling-scorekeep",
            "FixedRate": 0.003,
            "ReservoirQuota": 0,
            "ReservoirQuotaTTL": 1530923107.0,
            "Interval": 10
        }
    ],
    "LastRuleModification": 1530920505.0,
    "UnprocessedStatistics": []
}
```

来自 X-Ray 的响应包含要使用的配额（而不是从容器借用）。在此示例中，该服务在 10 秒钟内从容器借用了 10 条跟踪，并对其他 100 个请求应用了 10% 的固定速率，结果共有 20 个采样请求。配额有效期为五分钟（按生存时间表示），或者直到分配新的配额为止。X-Ray 也可能指定比默认报告间隔更长的间隔，尽管这里没有这样做。

> **Note**
>
> 来自 X-Ray 的响应可能不包含您首次调用它时的配额。继续从容器借用，直到为您分配配额。

响应中的其他两个字段可能表示输入有问题。请针对上一次 GetSamplingRules 调用检查 LastRuleModification。如果较新，则获取相应规则的新副本。UnprocessedStatistics 可以包括指示规则已删除、输入中的统计文档太旧或权限错误的错误。

AWS X-Ray 分段文档

跟踪分段是应用程序提供服务的请求的 JSON 表示形式。跟踪段记录有关原始请求的信息、有关您的应用程序在本地所做工作的信息，以及包含有关您的应用程序对 AWS 资源 APIs、HTTP 和 SQL 数据库进行的下游调用的信息的子段。

分段文档将有关分段的信息传递给 X-Ray。分段文档最多可为 64KB，并且包含一个带子分段的完整分段、指示请求正在进行中的分段部分或一个单独发送的子分段。您可以使用 PutTraceSegments API 将分段文档直接发送到 X-Ray。

X-Ray 编译和处理分段文档，生成可查询的跟踪摘要和完整轨迹，您可以分别使用GetTraceSummaries和BatchGetTraces APIs访问这些摘要。除了您发送到 X-Ray 的分段和子分段之外，服务还使用子分段中的信息生成推断分段并将其添加到完整跟踪。推断分段表示跟踪地图中的下游服务和资源。

X-Ray 提供分段文档的 JSON 架构。你可以在这里下载架构：xray-segmentdocument-schema-v1.0.0。以下部分中更详细地描述了该架构中列出的字段和对象。

X-Ray 为分段字段的子集编制索引以用于筛选表达式。例如，如果您将分段上的 user 字段设置为唯一标识符，则可在 X-Ray 控制台中或使用 GetTraceSummaries API 搜索与特定用户关联的分段。有关更多信息，请参阅 使用筛选条件表达式。

在使用 X-Ray SDK 检测应用程序时，此 SDK 将为您生成分段文档。此 SDK 通过本地 UDP 端口将分段文档传输到 X-Ray 进程守护程序，而不是直接将分段文档发送到 X-Ray。有关更多信息，请参阅 将分段文档发送到 X-Ray 进程守护程序。

Sections

- 分段字段
- 子分段
- HTTP 请求数据
- Annotations
- 元数据
- AWS 资源数据

- 错误和异常
- SQL 查询

分段字段

分段记录有关应用程序提供服务的请求的跟踪信息。分段至少会记录请求的名称、ID、开始时间、跟踪 ID 和结束时间。

Example 最小完成分段

```
{
  "name" : "example.com",
  "id" : "70de5b6f19ff9a0a",
  "start_time" : 1.478293361271E9,
  "trace_id" : "1-581cf771-a006649127e371903a2de979",
  "end_time" : 1.478293361449E9
}
```

分段需要或有条件地需要以下字段。

> **Note**
>
> 除非另行说明，否则值必须是字符串（最多 250 个字符）。

必填分段字段

- name - 处理了请求的服务的逻辑名称（最多 200 个字符）。例如，您的应用程序的名称或域名。名称可以包含 Unicode 字母、数字、空格和以下符号：_、 .、 :、 /、 %、 &、 #、 =、 +、 \、 -、 @
- id - 分段的 64 位标识符，在同一个跟踪中的分段之间唯一，使用 16 位十六进制数。
- trace_id - 连接源自单个客户端请求的所有分段和子分段的唯一标识符。

  X-Ray 跟踪 ID 格式

  X-Ray trace_id 由以连字符分隔的三组数字组成。例如，1-58406520-a006649127e371903a2de979。这包括：

  - 版本号，即 1。
  - 原始请求的时间，采用 Unix 纪元时间，为 8 个十六进制数字。

例如，2016 年 12 月 1 日上午 10:00（太平洋标准时间）的纪元时间为 1480615200 秒，或者是十六进制数字 58406520。

- 跟踪的 96 位全局唯一标识符，使用 24 个十六进制数字。

> **Note**
>
> X-Ray 现在支持使用创建的跟踪 IDs OpenTelemetry 以及任何其他符合 W3C 跟踪上下文规范的框架。发送到 X-Ray 时，W3C 跟踪 ID 必须采用 X-Ray 跟踪 ID 的格式。例如，W3C 跟踪 ID 4efaaf4d1e8720b39541901950019ee5 在发送到 X-Ray 时，应与 1-4efaaf4d-1e8720b39541901950019ee5 的格式相同。X-Ray 跟踪 IDs 包括以 Unix 纪元时间为单位的原始请求时间戳，但是当以 IDs X-Ray 格式发送 W3C 跟踪时，这不是必需的。

> **跟踪 ID 安全性**
>
> 跟踪 IDs 在响应标头中可见。使用安全的随机算法生成 IDs 跟踪，确保攻击者无法计算 future 的跟踪 IDs ，也无法使用这些跟踪 IDs 向您的应用程序发送请求。

- `start_time` - 表示分段的创建时间的数字，采用浮点秒数的纪元时间。例如，`1480615200.010` 或 `1.480615200010E9`。使用所需数量的小数位。建议使用微秒解析（如果可用）。
- `end_time` - 表示分段的关闭时间的数字。例如，`1480615200.090` 或 `1.480615200090E9`。指定 `end_time` 或 `in_progress`。
- `in_progress` - 布尔值`true`，设置为 ，而不是指定 `end_time` 以记录分段已经启动但未完成。在您的应用程序接收到需要长时间提供服务的请求时发送进行中的分段，用于跟踪请求接收。发送了响应之后，发送完成分段以覆盖进行中分段。仅为每个请求发送一个完整分段，以及一个或零个进行中分段。

> **服务名称**
>
> 分段的 `name` 应该与生成该分段的服务的域名或逻辑名称相匹配。但是，并未强制执行此规则。任何拥有 `PutTraceSegments` 权限的应用程序均可发送任何名称的分段。

以下字段是分段的可选字段。

可选分段字段

- service - 一个包含应用程序的相关信息的对象。
  - version - 一个标识为请求提供服务的应用程序版本的字符串。
- user - 一个标识发送请求的用户的字符串。
- origin— 运行应用程序的 AWS 资源类型。

  支持的值

  - AWS::EC2::Instance— 亚马逊 EC2 实例。
  - AWS::ECS::Container — 一个 Amazon ECS 容器。
  - AWS::ElasticBeanstalk::Environment - 一个 Elastic Beanstalk 环境。

  如果您的应用程序适用多个值，请使用最具体的值。例如，多容器 Docker Elastic Beanstalk 环境在亚马逊 ECS 容器上运行您的应用程序，而该容器又在亚马逊实例上运行。 EC2 在这种情况下，您应将源设为 AWS::ElasticBeanstalk::Environment，因为环境是另外两种资源的父级。
- parent_id - 您在请求源自检测过的应用程序时指定的子分段 ID。X-Ray SDK 将父级子分段 ID 添加到下游 HTTP 调用的跟踪标头。对于嵌套子分段，一个子分段可以有一个分段或一个子分段作为其父级。
- http - http 对象，包含原始 HTTP 请求的相关信息。
- aws— 包含有关您的应用程序为请求提供服务的 AWS 资源信息的aws对象。
- error、throttle、fault 和 cause - 错误字段，指示出现错误并且包含有关导致错误的异常的信息。
- annotations - annotations 对象，包含您希望 X-Ray 为其编制索引以进行搜索的键-值对。
- metadata - metadata 对象，包含要存储在分段中的任何附加数据。
- subsegments - subsegment 对象数组。

子分段

您可以创建子分段来记录对 AWS SDK 的调用 AWS 服务 和资源、对内部或外部 HTTP Web APIs 的调用或 SQL 数据库查询。您也可以创建子分段来调试应用程序中的代码块或为其添加注释。由于子分段可以包含其他子分段，因此，记录有关内部函数调用的元数据的自定义子分段可以包含其他自定义子分段和下游调用的子分段。

子分段从调用它的服务的角度，记录下游调用。X-Ray 使用子分段来确定未发送分段的下游服务，并在服务图上为其创建条目。

子分段可以嵌入到完整分段文档或者单独发送。对于长时间运行的请求，单独发送子分段以异步跟踪下游调用，或者避免超过最大分段文档大小。

Example 带有嵌入式子分段的分段

独立子分段具有 type 的 subsegment 以及标识父分段的 parent_id。

```
{
  "trace_id"   : "1-5759e988-bd862e3fe1be46a994272793",
  "id"         : "defdfd9912dc5a56",
  "start_time" : 1461096053.37518,
  "end_time"   : 1461096053.4042,
  "name"       : "www.example.com",
  "http"       : {
    "request"  : {
      "url"        : "https://www.example.com/health",
      "method"     : "GET",
      "user_agent" : "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6)
 AppleWebKit/601.7.7",
      "client_ip"  : "11.0.3.111"
    },
    "response" : {
      "status"         : 200,
      "content_length" : 86
    }
  },
  "subsegments" : [
    {
      "id"         : "53995c3f42cd8ad8",
      "name"       : "api.example.com",
      "start_time" : 1461096053.37769,
      "end_time"   : 1461096053.40379,
      "namespace"  : "remote",
      "http"       : {
        "request"  : {
          "url"    : "https://api.example.com/health",
          "method" : "POST",
          "traced" : true
        },
        "response" : {
          "status"         : 200,
          "content_length" : 861
        }
      }
```

```
    }
  ]
}
```

对于长时间运行的请求，您可以发送进行中分段来告知 X-Ray 已收到请求，然后在完成原始请求之前单独发送子分段来跟踪这些请求。

Example 正在进行分段

```
{
  "name" : "example.com",
  "id" : "70de5b6f19ff9a0b",
  "start_time" : 1.478293361271E9,
  "trace_id" : "1-581cf771-a006649127e371903a2de979",
  "in_progress": true
}
```

Example 独立子分段

独立子分段具有 `type` 的 `subsegment`，一个 `trace_id` 和一个标识父分段的 `parent_id`。

```
{
  "name" : "api.example.com",
  "id" : "53995c3f42cd8ad8",
  "start_time" : 1.478293361271E9,
  "end_time" : 1.478293361449E9,
  "type" : "subsegment",
  "trace_id" : "1-581cf771-a006649127e371903a2de979"
  "parent_id" : "defdfd9912dc5a56",
  "namespace"  : "remote",
  "http"       : {
      "request"  : {
          "url"    : "https://api.example.com/health",
          "method" : "POST",
          "traced" : true
      },
      "response" : {
          "status"         : 200,
          "content_length" : 861
      }
  }
}
```

在请求完成时，请使用 end_time 重新发送分段来关闭分段。完成分段将覆盖进行中分段。

您也可以为触发了异步工作流的已完成请求单独发送子分段。例如，在开始用户请求的工作之前，Web API 可能立即返回 OK 200 响应。您可以在发送响应后立即将完整分段发送到 X-Ray，然后为稍后完成的工作发送子分段。与分段一样，您还可以发送子分段片段来记录子分段已开始，然后在下游调用完成后用一个完整子分段覆盖此子分段。

子分段需要或有条件地需要以下字段。

> **Note**
>
> 除非另行说明，否则值是字符串（最多 250 个字符）。

必填子分段字段

- id - 子分段的 64 位标识符，在同一个跟踪中的分段之间唯一，使用 16 位十六进制数。
- name - 子分段的逻辑名称。对于下游调用，命名调用的资源或服务后的子分段。对于自定义子分段，命名其检测的代码后的子分段（例如，函数名称）。
- start_time - 表示创建子分段的时间的数字，采用浮点秒数的纪元时间，精确到毫秒。例如，1480615200.010 或 1.480615200010E9。
- end_time - 表示子分段的关闭时间的数字。例如，1480615200.090 或 1.480615200090E9。指定 end_time 或 in_progress。
- in_progress - 设置为 true 的布尔值，而不是指定 end_time 以记录子分段已经启动但未完成。仅为每个下游请求发送一个完整子分段，以及一个或零个进行中子分段。
- trace_id - 子分段的父分段的跟踪 ID。仅在单独发送子分段时是必需的。

  X-Ray 跟踪 ID 格式

  X-Ray trace_id 由以连字符分隔的三组数字组成。例如，1-58406520-a006649127e371903a2de979。这包括：

  - 版本号，即 1。
  - 原始请求的时间，采用 Unix 纪元时间，为 8 个十六进制数字。

    例如，2016 年 12 月 1 日上午 10:00（太平洋标准时间）的纪元时间为 1480615200 秒，或者是十六进制数字 58406520。
  - 跟踪的 96 位全局唯一标识符，使用 24 个十六进制数字。

> **Note**
>
> X-Ray 现在支持使用创建的跟踪 IDs OpenTelemetry 以及任何其他符合 W3C 跟踪上下文规范的框架。发送到 X-Ray 时，W3C 跟踪 ID 必须采用 X-Ray 跟踪 ID 的格式。例如，W3C 跟踪 ID `4efaaf4d1e8720b39541901950019ee5` 在发送到 X-Ray 时，应与 `1-4efaaf4d-1e8720b39541901950019ee5` 的格式相同。X-Ray 跟踪 IDs 包括以 Unix 纪元时间为单位的原始请求时间戳，但是当以 IDs X-Ray 格式发送 W3C 跟踪时，这不是必需的。

- `parent_id` - 子分段的父分段的分段 ID。仅在单独发送子分段时是必需的。对于嵌套子分段，一个子分段可以有一个分段或一个子分段作为其父级。
- `type` - `subsegment`。仅在单独发送子分段时是必需的。

以下字段是子分段的可选字段。

可选子分段字段

- `namespace` - 对于 AWS SDK 调用，为 `aws`；对于其他下游调用，为 `remote`。
- `http` - `http` 对象，包含有关传出 HTTP 调用的信息。
- `aws`— 包含有关您的应用程序调用的下游 AWS 资源的信息的`aws`对象。
- `error`、`throttle`、`fault` 和 `cause` - 错误字段，指示出现错误并且包含有关导致错误的异常的信息。
- `annotations` - `annotations` 对象，包含您希望 X-Ray 为其编制索引以进行搜索的键-值对。
- `metadata` - `metadata` 对象，包含要存储在分段中的任何附加数据。
- `subsegments` - `subsegment` 对象数组。
- `precursor_ids`— 子分段数组 IDs ，用于标识子分段，其父分段与该子分段之前完成的子分段相同。

HTTP 请求数据

使用 HTTP 数据块记录有关应用程序提供服务的 HTTP 请求（在分段中）或应用程序向下游 HTTP API 发出的 HTTP 请求（在子分段中）的详细信息。此对象中的大多数字段将映射到在 HTTP 请求和响应中找到的信息。

http

所有字段都是可选字段。

- request - 有关请求的信息。
  - method - 请求方法。例如，GET。
  - url - 从请求的协议、主机名和路径编译的完整请求 URL。
  - user_agent - 来自请求者客户端的用户代理字符串。
  - client_ip - 请求者的 IP 地址。可从 IP 数据包的 Source Address 或（对于转发的请求）X-Forwarded-For 标头中检索。
  - x_forwarded_for - （仅分段）布尔值，指示已从 X-Forwarded-For 标头读取 client_ip，并且它不可靠，因为它可能是伪造的。
  - traced - （仅子分段）布尔值，指示下游调用针对的是另一个跟踪的服务。如果此字段设置为 true，则 X-Ray 会将跟踪视为已中断，直至下游服务上传一个分段，此分段的 parent_id 与包含此数据块的子分段的 id 匹配。
- response - 有关响应的信息。
  - status - 指示响应的 HTTP 状态的整数。
  - content_length - 指示响应正文长度（以字节为单位）的整数。

在检测对下游 Web API 进行的调用时，记录包含有关 HTTP 请求和响应的信息的子分段。X-Ray 使用子分段为远程 API 生成推断分段。

Example 由在 Amazon 上运行的应用程序提供的 HTTP 调用的分段 EC2

```
{
  "id": "6b55dcc497934f1a",
  "start_time": 1484789387.126,
  "end_time": 1484789387.535,
  "trace_id": "1-5880168b-fd5158284b67678a3bb5a78c",
  "name": "www.example.com",
  "origin": "AWS::EC2::Instance",
  "aws": {
    "ec2": {
      "availability_zone": "us-west-2c",
      "instance_id": "i-0b5a4678fc325bg98"
    },
    "xray": {
        "sdk_version": "2.11.0 for Java"
```

```
    },
  },
  "http": {
    "request": {
      "method": "POST",
      "client_ip": "78.255.233.48",
      "url": "http://www.example.com/api/user",
      "user_agent": "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:45.0) Gecko/20100101
 Firefox/45.0",
      "x_forwarded_for": true
    },
    "response": {
      "status": 200
    }
  }
```

Example 下游 HTTP 调用的子分段

```
{
  "id": "004f72be19cddc2a",
  "start_time": 1484786387.131,
  "end_time": 1484786387.501,
  "name": "names.example.com",
  "namespace": "remote",
  "http": {
    "request": {
      "method": "GET",
      "url": "https://names.example.com/"
    },
    "response": {
      "content_length": -1,
      "status": 200
    }
  }
}
```

Example 下游 HTTP 调用的推断分段

```
{
  "id": "168416dc2ea97781",
  "name": "names.example.com",
  "trace_id": "1-62be1272-1b71c4274f39f122afa64eab",
  "start_time": 1484786387.131,
```

```
  "end_time": 1484786387.501,
  "parent_id": "004f72be19cddc2a",
  "http": {
    "request": {
      "method": "GET",
      "url": "https://names.example.com/"
    },
    "response": {
      "content_length": -1,
      "status": 200
    }
  },
  "inferred": true
}
```

Annotations

分段和子分段可包含一个 annotations 对象，此对象包含一个或多个字段，X-Ray 将为这些字段编制索引以便用于筛选表达式。字段可以包含字符串、数字或布尔值（无对象或数组）。X-Ray 最多为每个跟踪的 50 条注释编制索引。

Example 包含注释的 HTTP 调用的分段

```
{
  "id": "6b55dcc497932f1a",
  "start_time": 1484789187.126,
  "end_time": 1484789187.535,
  "trace_id": "1-5880168b-fd515828bs07678a3bb5a78c",
  "name": "www.example.com",
  "origin": "AWS::EC2::Instance",
  "aws": {
    "ec2": {
      "availability_zone": "us-west-2c",
      "instance_id": "i-0b5a4678fc325bg98"
    },
    "xray": {
        "sdk_version": "2.11.0 for Java"
    },
  },
  "annotations": {
    "customer_category" : 124,
    "zip_code" : 98101,
    "country" : "United States",
```

```
    "internal" : false
  },
  "http": {
    "request": {
      "method": "POST",
      "client_ip": "78.255.233.48",
      "url": "http://www.example.com/api/user",
      "user_agent": "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:45.0) Gecko/20100101
 Firefox/45.0",
      "x_forwarded_for": true
    },
    "response": {
      "status": 200
    }
  }
```

键必须为字母数字才能用于筛选器。允许使用下划线。不允许使用其他符号和空格。

元数据

分段和子分段可包含一个 metadata 对象，此对象包含一个或多个字段，这些字段具有任何类型的值（包括对象和数组）。X-Ray 不会为元数据编制索引，并且值可以是任何大小，前提是分段文档不会超出最大大小 (64KB)。您可以查看由 BatchGetTraces API 返回的完整分段文档中的元数据。以开头的字段键（debug在以下示例中）保留 AWS供-provided SDKs 和客户端使用。AWS.

Example 包含元数据的自定义子分段

```
{
  "id": "0e58d2918e9038e8",
  "start_time": 1484789387.502,
  "end_time": 1484789387.534,
  "name": "## UserModel.saveUser",
  "metadata": {
    "debug": {
      "test": "Metadata string from UserModel.saveUser"
    }
  },
  "subsegments": [
    {
      "id": "0f910026178b71eb",
      "start_time": 1484789387.502,
      "end_time": 1484789387.534,
      "name": "DynamoDB",
```

```
      "namespace": "aws",
      "http": {
        "response": {
          "content_length": 58,
          "status": 200
        }
      },
      "aws": {
        "table_name": "scorekeep-user",
        "operation": "UpdateItem",
        "request_id": "3AIENM5J4ELQ3SPODHKBIRVIC3VV4KQNSO5AEMVJF66Q9ASUAAJG",
        "resource_names": [
          "scorekeep-user"
        ]
      }
    }
  ]
}
```

AWS 资源数据

对于分段，aws 对象包含有关应用程序运行于的资源的信息。多个字段可应用于一个资源。例如，在 Elastic Beanstalk 的多容器 Docker 环境中运行的应用程序可能包含有关亚马逊实例、该实例上运行的 A EC2 mazon ECS 容器以及 Elastic Beanstalk 环境本身的信息。

aws (分段)

所有字段都是可选字段。

- `account_id`— 如果您的应用程序向其他应用程序发送区段 AWS 账户，请记录运行您的应用程序的账户的 ID。
- `cloudwatch_logs`— 描述单个 CloudWatch 日志组的对象数组。
  - `log_group`— CloudWatch 日志组名称。
  - `arn`— CloudWatch 日志组 ARN。
- `ec2`— 有关 Amazon EC2 实例的信息。
  - `instance_id`— 实例的实 EC2 例 ID。
  - `instance_size`— EC2 实例的类型。
  - `ami_id`— Amazon 系统映像 ID。
  - `availability_zone` - 实例在其中运行的可用区。

- ecs - 有关 Amazon ECS 实例的信息。
  - container— 您的容器的主机名。
  - container_id— 您的容器的完整容器 ID。
  - container_arn - 容器实例的 ARN。
- eks - 有关 Amazon EKS 集群的信息。
  - pod— EKS 容器组的主机名。
  - cluster_name - EKS 集群名称。
  - container_id— 您的容器的完整容器 ID。
- elastic_beanstalk— 有关 Elastic Beanstalk 环境的信息。您可以在最新 Elastic Beanstalk 平台上名为 /var/elasticbeanstalk/xray/environment.conf 的文件中找到该信息。
  - environment_name – 环境名称。
  - version_label - 当前部署到为请求提供服务的实例的应用程序版本的名称。
  - deployment_id - 数字，指示针对为请求提供服务的实例的上次成功部署的 ID。
- xray – 有关所使用检测的类型和版本的元数据。
  - auto_instrumentation – 布尔值，指示是否使用了自动检测（例如，Java 代理）。
  - sdk_version - 正在使用的 SDK 或代理的版本。
  - sdk - SDK 类型。

Example AWS 用插件屏蔽

```
"aws":{
   "elastic_beanstalk":{
      "version_label":"app-5a56-170119_190650-stage-170119_190650",
      "deployment_id":32,
      "environment_name":"scorekeep"
   },
   "ec2":{
      "availability_zone":"us-west-2c",
      "instance_id":"i-075ad396f12bc325a",
      "ami_id":
   },
   "cloudwatch_logs":[
      {
         "log_group":"my-cw-log-group",
         "arn":"arn:aws:logs:us-west-2:012345678912:log-group:my-cw-log-group"
      }
```

```
    ],
    "xray":{
       "auto_instrumentation":false,
       "sdk":"X-Ray for Java",
       "sdk_version":"2.8.0"
    }
}
```

对于子分段，请记录有关您的应用程序访问的 AWS 服务 和资源的信息。X-Ray 使用此信息来创建推断分段，这些分段表示服务地图中的下游服务。

aws (子分段)

所有字段都是可选字段。

- `operation`— 针对 AWS 服务 或资源调用的 API 操作的名称。
- `account_id`— 如果您的应用程序访问其他账户中的资源，或者向其他账户发送分段，请记录拥有您的应用程序所访问 AWS 资源的账户的 ID。
- `region` - 如果资源所在的区域不同于应用程序所在的区域，则记录前者。例如，`us-west-2`。
- `request_id` - 请求的唯一标识符。
- `queue_url` - 对于 Amazon SQS 队列上的操作，为队列的 URL。
- `table_name` - 对于 DynamoDB 表上的操作，为表的名称。

Example 对 DynamoDB 进行调用以保存项目的子分段

```
{
  "id": "24756640c0d0978a",
  "start_time": 1.480305974194E9,
  "end_time": 1.4803059742E9,
  "name": "DynamoDB",
  "namespace": "aws",
  "http": {
    "response": {
      "content_length": 60,
      "status": 200
    }
  },
  "aws": {
    "table_name": "scorekeep-user",
    "operation": "UpdateItem",
```

```
    "request_id": "UBQNSO5AEM8T4FDA4RQDEB94OVTDRVV4K4HIRGVJF66Q9ASUAAJG",
  }
}
```

错误和异常

出错时，您可以记录有关该错误及其生成的异常的详细信息。当应用程序将错误返回给用户时，在分段中记录错误；当下游调用返回错误时，在子分段中记录错误。

错误类型

将以下一个或多个字段设置为 `true` 可指示已发生错误。如果出现复合错误，则多个类型适用。例如，来自下游调用的 `429 Too Many Requests` 错误可能会导致应用程序返回 `500 Internal Server Error`，在此情况下，所有三种类型将适用。

- `error` - 布尔值，指示出现客户端错误（响应状态代码为 4XX 客户端错误）。
- `throttle` - 布尔值，指示请求已受限（响应状态代码为 429 请求过多）。
- `fault` - 布尔值，指示出现服务器错误（响应状态代码为 5XX 服务器错误）。

通过在分段或子分段中包含 cause 对象来指示错误原因。

`cause`

原因可以是 16 个字符的异常 ID 或带以下字段的对象：

- `working_directory` - 发生异常时的工作目录的完整路径。
- `paths` - 发生异常时所使用的库或模块的路径的数组。
- `exceptions` - 异常对象的数组。

包含有关一个或多个 exception 对象中的错误的详细信息。

`exception`

所有字段都是可选字段。

- `id` - 异常的 64 位标识符，在同一个跟踪中的分段之间唯一，使用 16 位十六进制数。
- `message` - 异常消息。
- `type` - 异常类型。

- `remote` - 布尔值，指示由下游服务返回的错误导致的异常。
- `truncated` - 整数，指示从 `stack` 中忽略的堆栈帧数。
- `skipped` - 整数，指示在此异常与其子异常（此异常导致的异常）之间跳过的异常数。
- `cause` - 此异常的父级（导致此异常的异常）的异常 ID。
- `stack` - stackFrame 对象的数组。

如果可用，则记录有关 stackFrame 对象中的调用堆栈的信息。

`stackFrame`

所有字段都是可选字段。

- `path` - 文件的相对路径。
- `line` - 文件中的行。
- `label` - 函数或方法名称。

SQL 查询

您可以为应用程序向 SQL 数据库发出的查询创建子分段。

`sql`

所有字段都是可选字段。

- `connection_string` - 对于 SQL Server 连接或不使用 URL 连接字符串的其他数据库连接，记录连接字符串（不包括密码）。
- `url` - 对于使用 URL 连接字符串的数据库连接，记录 URL（不包括密码）。
- `sanitized_query` - 数据库查询，其任何用户提供的值已删除或由占位符替换。
- `database_type` - 数据库引擎的名称。
- `database_version` - 数据库引擎的版本号。
- `driver_version` - 应用程序使用的数据库引擎驱动程序的名称和版本号。
- `user` - 数据库用户名。
- `preparation` - 如果查询使用了 `PreparedCall`，则为 `call`；如果查询使用了 `PreparedStatement`，则为 `statement`。

Example 具有 SQL 查询的子分段

```
{
  "id": "3fd8634e78ca9560",
  "start_time": 1484872218.696,
  "end_time": 1484872218.697,
  "name": "ebdb@aawijb5u25wdoy.cpamxznpdoq8.us-west-2.rds.amazonaws.com",
  "namespace": "remote",
  "sql" : {
    "url": "jdbc:postgresql://aawijb5u25wdoy.cpamxznpdoq8.us-
west-2.rds.amazonaws.com:5432/ebdb",
    "preparation": "statement",
    "database_type": "PostgreSQL",
    "database_version": "9.5.4",
    "driver_version": "PostgreSQL 9.4.1211.jre7",
    "user" : "dbuser",
    "sanitized_query" : "SELECT  *  FROM  customers  WHERE  customer_id=?;"
  }
}
```

AWS X-Ray 概念

AWS X-Ray 以分段形式接收来自服务的数据。然后，X-Ray 将具有共同请求的分段分组为跟踪。X-Ray 处理跟踪以生成服务图，服务图提供您的应用程序的可视化表示形式。

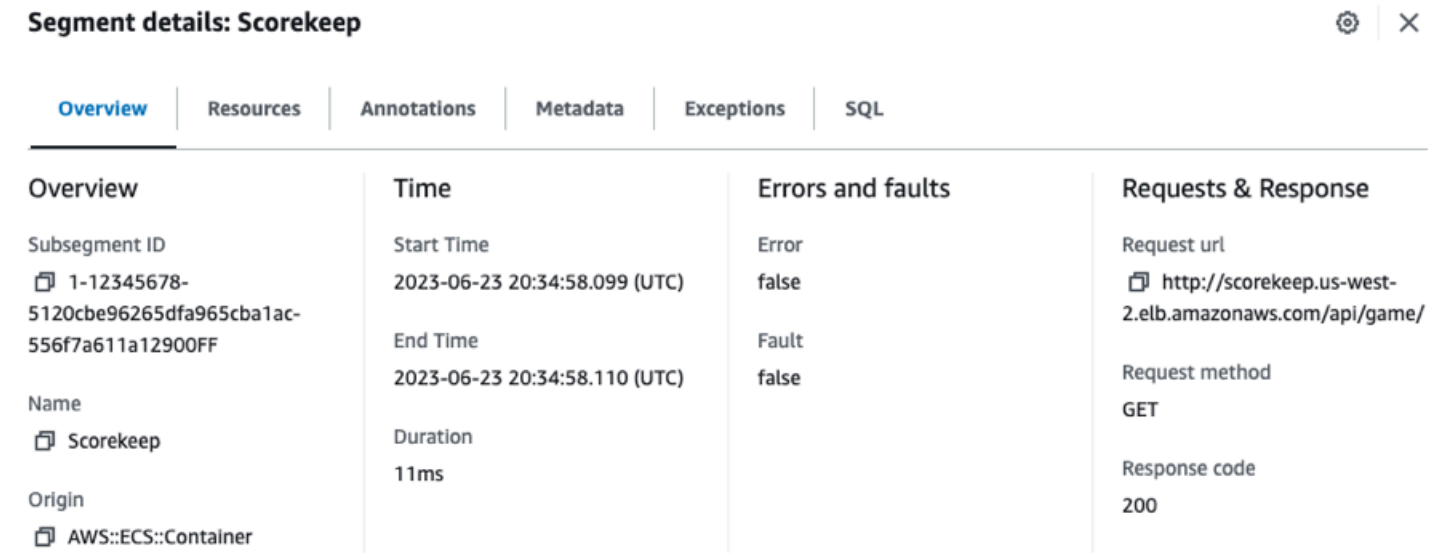
概念

- Segments
- 子分段
- 服务图
- 跟踪
- 采样
- 跟踪标头
- 筛选条件表达式
- 组
- 注释和元数据
- 错误、故障和异常

Segments

运行您的应用程序逻辑的计算资源发送关于其工作的数据作为分段。分段提供资源的名称、有关请求的详细信息以及有关所完成工作的详细信息。例如，当 HTTP 请求到达您的应用程序时，它可以记录下列相关数据：

- 主机 - 主机名、别名或 IP 地址
- 请求 - 方法，客户端地址、路径、用户代理
- 响应 - 状态、内容
- 所完成工作 - 开始和结束时间、子分段
- 发生的错误 - 错误、故障和异常，包括自动捕获的异常堆栈。



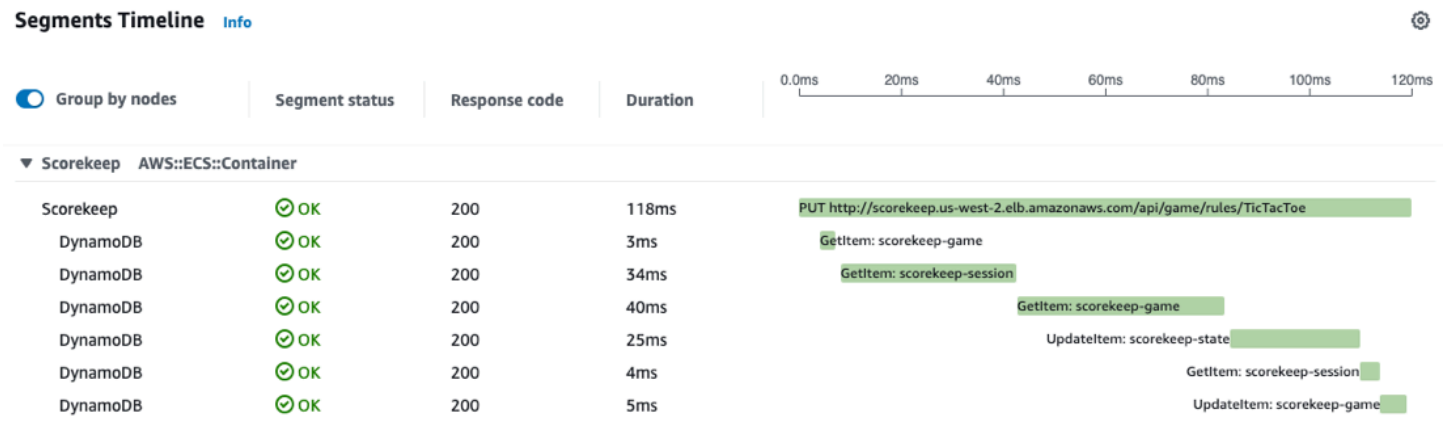
X-Ray SDK 从请求和响应标头、应用程序中的代码以及有关其运行 AWS 资源的元数据中收集信息。您可以通过修改应用程序配置或代码来选择要收集的数据，以检测传入的请求、下游请求和 AWS SDK 客户端。

> **转发的请求**
>
> 如果负载均衡器或其他中间将请求转发到您的应用程序，X-Ray 会提取请求 `X-Forwarded-For` 标头中的客户端 IP 而非 IP 数据包中的源 IP。由于转发的请求记录的客户端 IP 可以伪造，因此不应信任。

您可以使用 X-Ray SDK 来记录其他信息，如注释和元数据。有关分段和子分段中记录的结构和信息的详情，请参阅 AWS X-Ray 分段文档。分段文档的大小最大可以是 64KB。

子分段

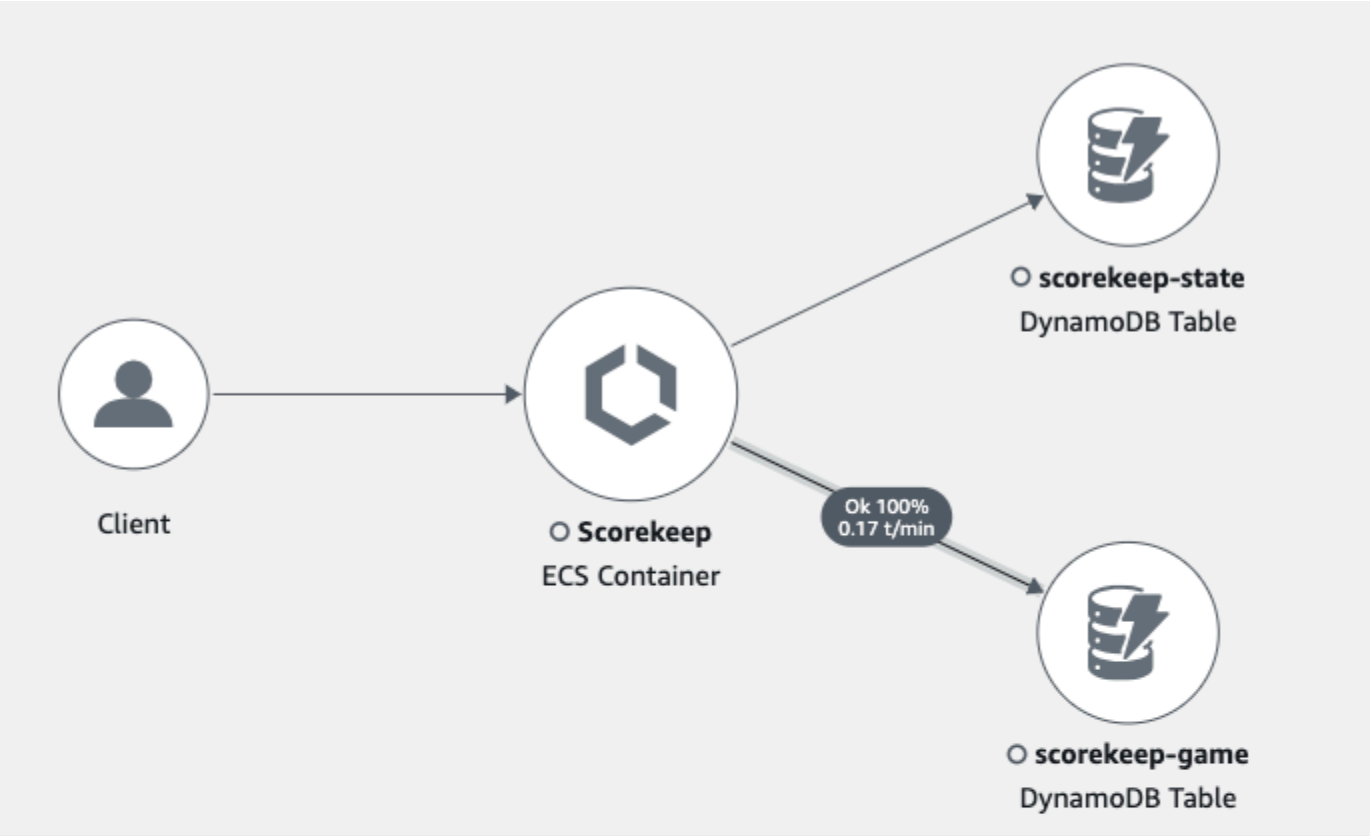
分段可以将关于已完成工作的数据细分为子分段。子分段提供有关您的应用程序为满足原始请求而进行的下游调用的更精细的计时信息和详情。子分段可以包含有关调用 AWS 服务、外部 HTTP API 或 SQL 数据库的更多详细信息。您甚至可以定义任意子分段以检测特定函数或应用程序中的代码行。

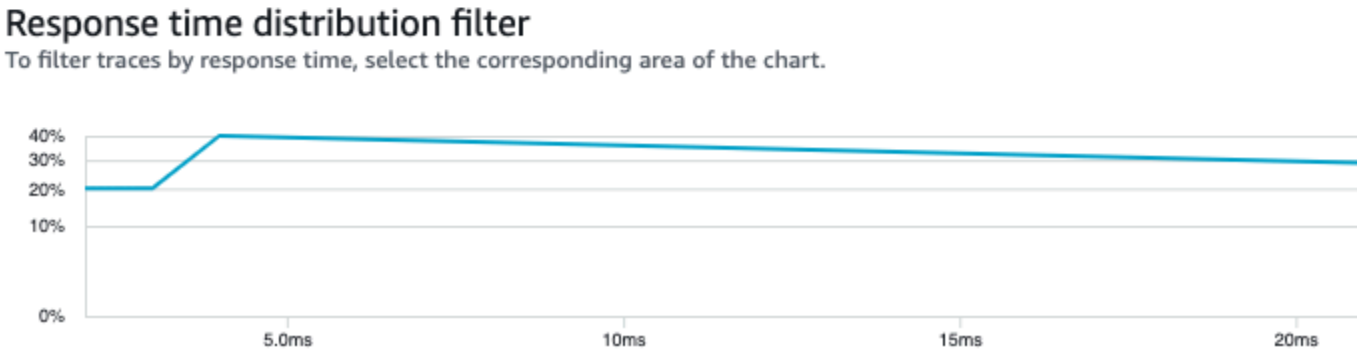


对于不发送自己的分段的服务（如 Amazon DynamoDB），X-Ray 使用子分段在跟踪地图上生成推断分段 和下游节点。这样您可以查看所有下游依赖项，即使它们不支持跟踪或者是外部依赖项也是如此。

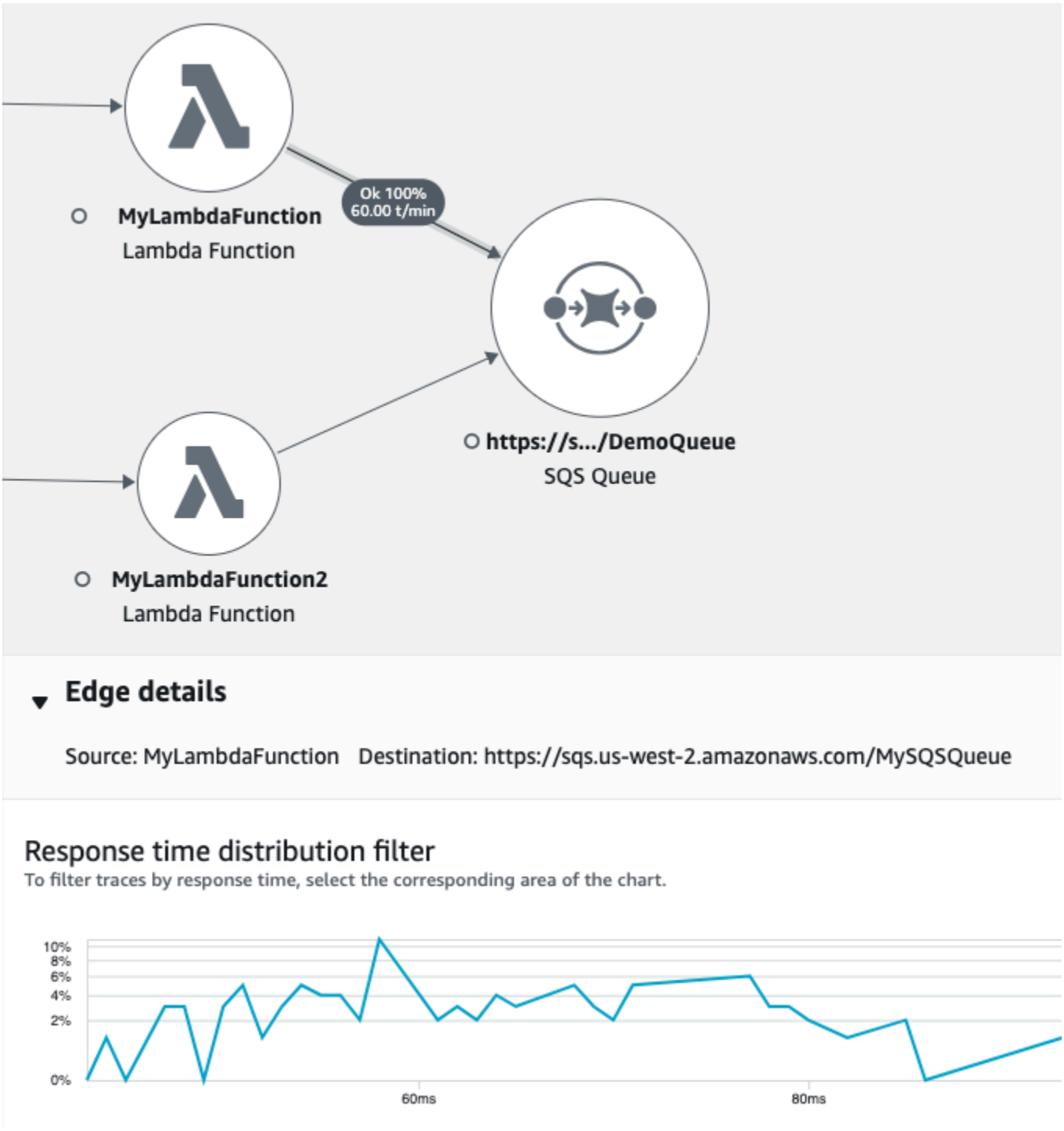
子分段表示从您应用程序的角度将下游调用视为客户端。如果还会检测下游服务，则它发送的分段会替换从上游客户端的子分段生成的推断分段。服务图上的节点使用来自服务分段的信息 (如果可用)，而两个节点之间的边缘节点使用上游服务的子分段。

例如，当您使用经过 AWS 检测的 SDK 客户端调用 DynamoDB 时，X-Ray SDK 会记录该调用的子分段。DynamoDB 不发送分段，因此跟踪中的推断分段、服务图上的 DynamoDB 节点以及您的服务与 DynamoDB 之间的边缘节点全都包含来自子分段的信息。





当您使用检测的应用程序调用另一个检测的服务时，下游服务会发送自己的分段，以从自己的角度记录上游服务在子分段中记录的相同调用。在服务图中，这两个服务的节点都包含来自这些服务的分段的计时和错误信息，而它们之间的边缘节点包含来自上游服务的分段的信息。



这两个视角都非常有用，因为下游服务精确地记录该应用程序开始和结束处理请求的时间，而上游服务记录往返延迟，包括在两个服务之间传输时请求所花费的时间。

服务图

X-Ray 使用您的应用程序发送的数据来生成服务图。向 X-Ray 发送数据的每个 AWS 资源都以服务形式出现在图表中。边缘连接协同工作以服务于请求的服务。边缘将客户端连接到您的应用程序，又将您的应用程序连接到它所使用的下游服务和资源。

> **服务名称**
>
> 分段的 name 应该与生成该分段的服务的域名或逻辑名称相匹配。但是，并未强制执行此规则。任何拥有 `PutTraceSegments` 权限的应用程序均可发送任何名称的分段。

服务图是一个 JSON 文档，其中包含有关构成您的应用程序的服务和资源的信息。X-Ray 控制台使用服务图来生成可视化形式或服务地图。

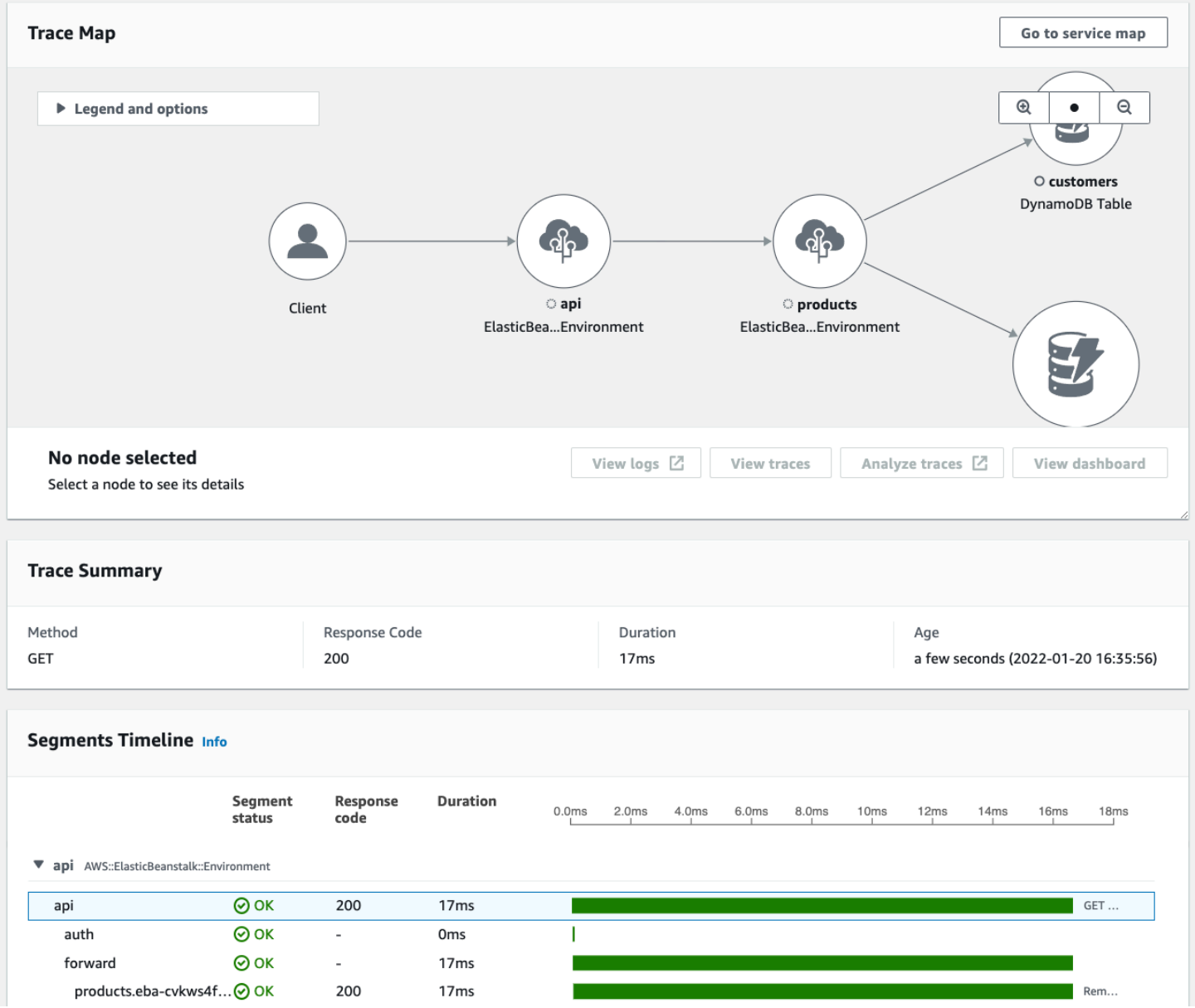
对于分布式应用程序，X-Ray 将处理具有相同跟踪 ID 的请求的服务的节点组合成一个服务图。请求命中的第一个服务会添加一个跟踪标头，该跟踪标头在前端及其所调用的服务之间传播。

例如，Scorekeep 运行一个调用微服务（ AWS Lambda 函数）的 Web API，通过使用 Node.js 库来生成随机名称。X-Ray SDK for Java 生成跟踪 ID 并将其包含在对 Lambda 的调用中。Lambda 发送跟踪数据并将跟踪 ID 传递给函数。X-Ray SDK for Node.js 还使用跟踪 ID 发送数据。结果，API、Lambda 服务和 Lambda 函数的节点在跟踪地图上全都显示为看似分离其实连接的节点。

服务图数据的保留期为 30 天。

跟踪

跟踪 ID 可跟踪请求通过您的应用程序的路径。跟踪会收集单个请求生成的所有分段。该请求通常是一个 HTTP GET 或 POST 请求，它通过负载均衡器传输，命中您的应用程序代码，并生成对其他 AWS 服务或外部 Web 的下游调用 APIs。HTTP 请求与之交互的第一个受支持服务将向请求中添加一个跟踪 ID 标头，并向下游传播该标头以跟踪延迟、处置和其他请求数据。



请参阅 AWS X-Ray 定价，了解 X-Ray 跟踪的计费方式。跟踪数据保留 30 天。

采样

为确保高效跟踪并为应用程序所服务的请求提供代表性样本，X-Ray SDK 应用采样算法来确定跟踪哪些请求。默认情况下，X-Ray 开发工具包每秒记录第一个请求，以及任何其他请求的百分之五。

为避免在您入门时产生服务费用，保守做法是使用默认采样率。您可以配置 X-Ray 以修改默认采样规则并配置基于服务或请求的属性应用采样的其他规则。

例如，您可能希望禁用采样，并跟踪对修改状态或处理用户或交易的调用的所有请求。对于量非常大的只读调用，例如后台轮询、运行状况检查或连接维护，您采用较低的采样率仍可获取足够的数据来了解出现的任何问题。

有关更多信息，请参阅 配置采样规则。

跟踪标头

所有请求都被跟踪，直到一个可配置的最低限度。在达到这一最低限度后，只有一部分请求被跟踪，以避免不必要的开销。采样决策和跟踪 ID 添加到名为 的跟踪标头`X-Amzn-Trace-Id`的 HTTP 请求中。请求命中的第一个 X-Ray-integrated服务会添加一个跟踪标头，该标头由 X-Ray SDK 读取并包含在响应中。

Example 具有根跟踪 ID 和采样决策的跟踪标头

```
X-Amzn-Trace-Id: Root=1-5759e988-
bd862e3fe1be46a994272793;Parent=53995c3f42cd8ad8;Sampled=1
```

> **跟踪标头安全性**
>
> 跟踪标头可以源自 X-Ray SDK AWS 服务、或客户端请求。您的应用程序可以`X-Amzn-Trace-Id`从传入的请求中删除，以避免用户在请求中添加跟踪 IDs 或采样决策而导致的问题。

如果请求来自检测的应用程序，跟踪标头还可以包含父分段 ID。例如，如果应用程序使用检测的 HTTP 客户端调用下游 HTTP Web API，则 X-Ray SDK 将原始请求的分段 ID 添加到下游请求的跟踪标头中。为下游请求提供服务的检测应用程序，可以记录父分段 ID 以连接两个请求。

Example 跟踪标头带有根跟踪 ID、父分段 ID 和采样决策

```
X-Amzn-Trace-Id: Root=1-5759e988-
bd862e3fe1be46a994272793;Parent=53995c3f42cd8ad8;Sampled=1
```

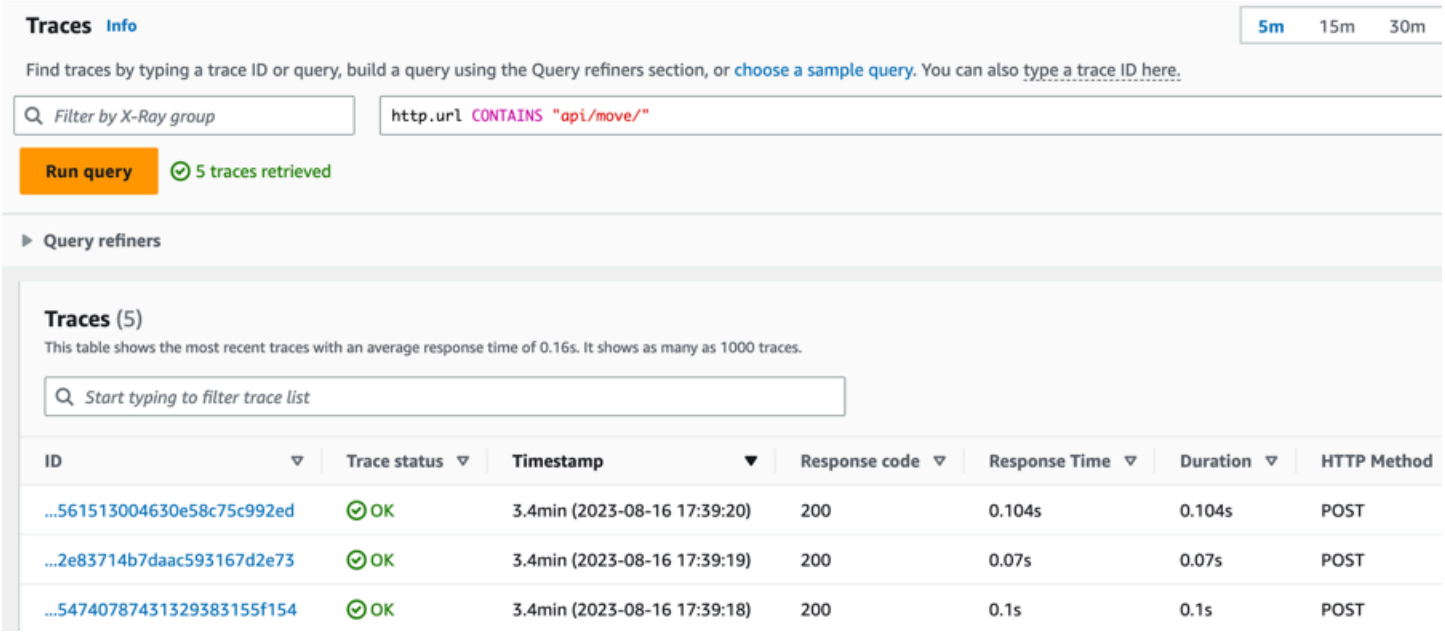
`Lineage`Lambda AWS 服务 和其他人可以将其附加到跟踪标头中，作为其处理机制的一部分，不应直接使用。

Example 跟踪带有世系的标头

```
X-Amzn-Trace-Id: Root=1-5759e988-
bd862e3fe1be46a994272793;Parent=53995c3f42cd8ad8;Sampled=1;Lineage=a87bd80c:1|
68fd508a:5|c512fbe3:2
```

筛选条件表达式

即使使用采样，复杂应用程序也会生成大量数据。 AWS X-Ray 控制台提供服务图的 easy-to-navigate 视图。它显示运行状况和性能信息，帮助您识别问题和机会，用于优化应用程序。对于高级跟踪，您可以细化以跟踪单个请求，或者使用筛选表达式来查找与特定路径或用户相关的跟踪。



组

通过扩展筛选条件表达式，X-Ray 也支持组功能。通过使用筛选条件表达式，您可以定义接受跟踪进入组的标准。

您可以按名称或按亚马逊资源名称 (ARN) 调用该组，以生成其自己的服务图表、跟踪摘要和亚马逊 CloudWatch 指标。创建组后，将根据组的筛选条件表达式检查传入跟踪，因为它们存储在 X-Ray 服务中。与每个条件匹配的跟踪数量的指标将发布到 CloudWatch 每分钟。

更新组的筛选条件表达式不会更改已记录的数据。更新仅应用于后续跟踪。这可能会生成新旧表达式的合并图。为避免发生这种情况，请删除当前群组并创建一个新的群组。

> **Note**
>
> 群组按检索到的符合筛选条件表达式的追踪数量计费。有关更多信息，请参阅 AWS X-Ray 定价。

有关组的更多信息，请参阅 配置组。

注释和元数据

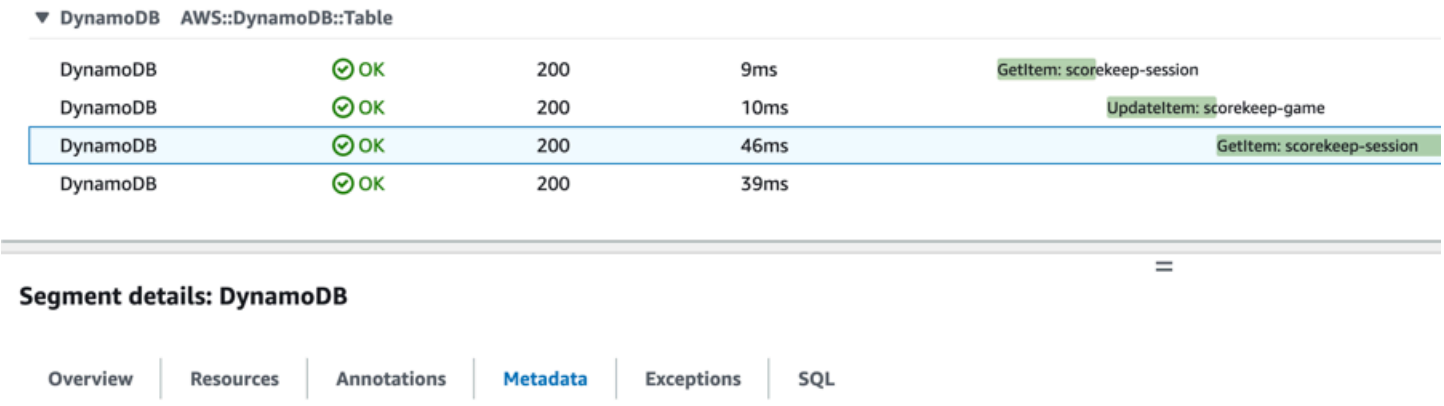
在检测应用程序时，X-Ray SDK 会记录有关传入和传出的请求、使用的 AWS 资源以及应用程序本身的信息。您可以向分段文档中添加其他信息作为注释和元数据。注释和元数据在跟踪级别汇总，可以添加到任何分段或子分段。

注释 是简单的键-值对，经编制索引后用于筛选条件表达式。使用注释记录要用于对控制台中的跟踪进行分组的数据或在调用 `GetTraceSummaries` API 时使用的数据。

X-Ray 最多为每个跟踪的 50 条注释编制索引。

元数据是具有任何类型值的键-值对，包括对象和列表，但没有编制索引。使用元数据记录要存储在跟踪中但不需要用于搜索跟踪的数据。

您可以在 CloudWatch 控制台的跟踪详细信息页面的区段或子区段详细信息窗口中查看注释和元数据。



错误、故障和异常

X-Ray 跟踪在您的应用程序代码中发生的错误以及下游服务返回的错误。错误分类如下。

- **`Error`** - 客户端错误（400 系列错误）

- **`Fault`** - 服务器故障（500 系列错误）
- **`Throttle`** - 限制错误（429 请求过多）

如果在您的应用程序为某个检测的请求提供服务时发生异常，X-Ray SDK 会记录有关异常的详细信息，包括堆栈跟踪（如果可用）。您可以在 X-Ray 控制台的分段详细信息下方查看异常。

安全性 AWS X-Ray

云安全 AWS 是重中之重。作为 AWS 客户，您可以从专为满足大多数安全敏感型组织的要求而构建的数据中心和网络架构中受益。

安全是双方共同承担 AWS 的责任。责任共担模式将其描述为云的 安全性和云中 的安全性：

- 云安全 — AWS 负责保护在云 AWS 服务 中运行的基础架构 AWS 云。 AWS 还为您提供可以安全使用的服务。作为 AWS 合规性计划的一部分，我们的安全措施的有效性定期由第三方审计员进行测试和验证。要了解适用于 X-Ray 的合规性计划，请参阅按合规性计划提供的范围内AWS 服务。
- 云端安全 — 您的责任由您 AWS 服务 使用的内容决定。您还需要对其他因素负责，包括您的数据的敏感性、您组织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 X-Ray 时应用责任共担模型。以下主题说明如何配置 X-Ray 以实现您的安全性和合规性目标。您还将学习如何使用其他 AWS 服务 方法来帮助您监控和保护您的 X-Ray 资源。

主题

中的数据保护 AWS X-Ray

AWS X-Ray 始终对静态跟踪和相关数据进行加密。当您需要审核和禁用加密密钥以满足合规性或内部要求时，可以将 X-Ray 配置为使用 AWS Key Management Service (AWS KMS) 密钥来加密数据。

X-Ray 提供了一个 AWS 托管式密钥 名字`aws/xray`。如果您只希望审核 AWS CloudTrail中的密钥使用情况，不需要管理密钥本身，请使用此密钥。如果您需要管理对密钥的访问权限或配置密钥轮换，可以创建客户托管密钥。

如果更改加密设置，X-Ray 需要花费一些时间来生成和传播数据密钥。在处理新密钥的过程中，X-Ray 可能会使用新旧设置的组合加密数据。更改加密设置后，不会对现有数据进行重新加密。

> **Note**
>
> AWS KMS 当 X-Ray 使用 KMS 密钥加密或解密跟踪数据时收费。
>
> - 默认加密 - 免费。
> - AWS 托管式密钥 — 付费使用密钥。
> - 客户托管密钥 - 存储和使用密钥需付费。
>
> 有关详细信息，请参阅AWS Key Management Service 定价。

> **Note**
>
> X-Ray insights 通知会将事件发送到亚马逊 EventBridge，而亚马逊目前不支持客户托管密钥。有关更多信息，请参阅 Amazon 中的数据保护 EventBridge。

您必须具有对客户托管密钥的用户级访问权限才能将 X-Ray 配置为使用该密钥然后查看加密的跟踪。请参阅用户加密权限了解更多信息。

CloudWatch console

使用 CloudWatch 控制台将 X-Ray 配置为使用 KMS 密钥进行加密

1. 登录 AWS Management Console 并打开 CloudWatch 控制台，网址为https://console.aws.amazon.com/cloudwatch/。
2. 在左侧导航窗格中，选择设置。
3. 在 X-Ray 跟踪部分中的加密下，选择查看设置。
4. 在加密配置部分，选择编辑。
5. 选择使用 KMS 密钥。
6. 从下拉菜单中选择一个密钥：
   - aws/xray – 使用 AWS 托管式密钥。
   - 密钥别名 - 在您的账户中使用客户托管 CMK。
   - 手动输入密钥 使用另一账户中的客户托管密钥。在出现的字段中输入密钥的完整 Amazon 资源名称 (ARN)。

7. 选择更新加密。

X-Ray console

使用 X-Ray 控制台将 X-Ray 配置为使用 KMS 密钥进行加密

1. 打开 X-Ray 控制台。
2. 选择加密。
3. 选择使用 KMS 密钥。
4. 从下拉菜单中选择一个密钥：
   - aws/xray – 使用 AWS 托管式密钥。
   - 密钥别名 - 在您的账户中使用客户托管 CMK。
   - 手动输入密钥 使用另一账户中的客户托管密钥。在出现的字段中输入密钥的完整 Amazon 资源名称 (ARN)。
5. 选择应用。

> **Note**
>
> X-Ray 不支持非对称 KMS 密钥。

如果 X-Ray 无法访问您的加密密钥，会停止存储数据。如果您的用户无法访问 KMS 密钥，或者您禁用了当前正在使用的密钥，会发生这种情况。如果发生这种情况，X-Ray 会在导航栏中显示了一个通知。

要使用 X-Ray API 配置加密设置，请参阅 利用 AWS X-Ray API 配置采样、组和加密设置。

的身份和访问管理 AWS X-Ray

AWS Identity and Access Management (IAM) AWS 服务 可帮助管理员安全地控制对 AWS 资源的访问权限。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 X-Ray 资源的人员。您可以使用 IAM AWS 服务 ，无需支付额外费用。

主题

- 受众

- 使用身份进行身份验证
- 使用策略管理访问
- 如何 AWS X-Ray 与 IAM 配合使用
- AWS X-Ray 基于身份的策略示例
- 对 AWS X-Ray 身份和访问进行故障排除

受众

您的使用方式 AWS Identity and Access Management (IAM) 会有所不同，具体取决于您在 X-Ray 中所做的工作。

服务用户 – 如果使用 X-Ray 服务来完成任务，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 X-Ray 特征来完成工作时，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 X-Ray 中的特征，请参阅 对 AWS X-Ray 身份和访问进行故障排除。

服务管理员 – 如果您在公司负责管理 X-Ray 资源，则您可能具有 X-Ray 的完全访问权限。您有责任确定您的服务用户应访问哪些 X-Ray 特征和资源。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 X-Ray 搭配使用的更多信息，请参阅 如何 AWS X-Ray 与 IAM 配合使用。

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 X-Ray 的访问权限的详细信息。要查看您可在 IAM 中使用的 X-Ray 基于身份的策略示例，请参阅 AWS X-Ray 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 AWS 使用身份凭证登录的方式。您必须以 IAM 用户身份或通过担 AWS 账户根用户任 IAM 角色进行身份验证（登录 AWS）。

您可以使用通过身份源提供的凭据以 AWS 联合身份登录。 AWS IAM Identity Center （IAM Identity Center）用户、贵公司的单点登录身份验证以及您的 Google 或 Facebook 凭据就是联合身份的示例。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。当你使用联合访问 AWS 时，你就是在间接扮演一个角色。

根据您的用户类型，您可以登录 AWS Management Console 或 AWS 访问门户。有关登录的更多信息 AWS，请参阅《AWS 登录 用户指南》中的如何登录到您 AWS 账户的。

如果您 AWS 以编程方式访问，则会 AWS 提供软件开发套件 (SDK) 和命令行接口 (CLI)，以便使用您的凭据对请求进行加密签名。如果您不使用 AWS 工具，则必须自己签署请求。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》中的用于签署 API 请求的AWS 签名版本 4。

无论使用何种身份验证方法，您都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 IAM 中的AWS 多重身份验证。

AWS 账户 root 用户

创建时 AWS 账户，首先要有一个登录身份，该身份可以完全访问账户中的所有资源 AWS 服务 和资源。此身份被称为 AWS 账户 root 用户，使用您创建账户时使用的电子邮件地址和密码登录即可访问该身份。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。有关需要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。

IAM 用户和群组

I AM 用户是您 AWS 账户 内部对个人或应用程序具有特定权限的身份。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用例，应在需要时更新访问密钥。

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，您可以拥有一个名为的群组，IAMAdmins并向该群组授予管理 IAM 资源的权限。

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 IAM 用户的使用案例。

IAM 角色

I AM 角色是您内部具有特定权限 AWS 账户 的身份。它类似于 IAM 用户，但与特定人员不关联。要在中临时担任 IAM 角色 AWS Management Console，您可以从用户切换到 IAM 角色（控制台）。您可以通过调用 AWS CLI 或 AWS API 操作或使用自定义 URL 来代入角色。有关使用角色的方法的更多信息，请参阅《IAM 用户指南》中的代入角色的方法。

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。有关用于联合身份验证的角色的信息，请参阅《IAM 用户指南》中的针对第三方身份提供商创建角色（联合身份验证）。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，对于某些资源 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 — 有些 AWS 服务 使用其他 AWS 服务服务中的功能。例如，当您在服务中拨打电话时，该服务通常会在 Amazon 中运行应用程序 EC2 或在 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 (FAS) — 当您使用 IAM 用户或角色在中执行操作时 AWS，您被视为委托人。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 使用调用委托人的权限以及 AWS 服务 向下游服务发出请求的请求。 AWS 服务只有当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项操作的权限。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-服务相关角色是一种链接到的服务角色。 AWS 服务服务可以代入代表您执行操作的角色。服务相关角色出现在您的中 AWS 账户 ，并且归服务所有。IAM 管理员可以查看但不能编辑服务相关角色的权限。
- 在 A@@ mazon 上运行的应用程序 EC2 — 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证书。这比在 EC2 实例中存储访问密钥更可取。要为 EC2 实例分配 AWS 角色并使其可供其所有应用程序使用，您需要创建一个附加到该实例的实例配置文件。实例配置文件包含角色并允许在 EC2 实例上运行的程序获得临时证书。有关更多信息，请参阅 IAM 用户指南中的使用 IAM 角色向在 A mazon EC2 实例上运行的应用程序授予权限。

使用策略管理访问

您可以 AWS 通过创建策略并将其附加到 AWS 身份或资源来控制中的访问权限。策略是其中的一个对象 AWS ，当与身份或资源关联时，它会定义其权限。 AWS 在委托人（用户、root 用户或角色会

话）发出请求时评估这些策略。策略中的权限确定是允许还是拒绝请求。大多数策略都以 JSON 文档的 AWS 形式存储在中。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。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。拥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》中的使用客户托管策略定义自定义 IAM 权限。

基于身份的策略可以进一步归类为内联策略或托管式策略。内联策略直接嵌入单个用户、组或角色中。托管策略是独立的策略，您可以将其附加到中的多个用户、群组和角色 AWS 账户。托管策略包括 AWS 托管策略和客户托管策略。要了解如何在托管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。委托人可以包括账户、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 IAM 中的 AWS 托管策略。

访问控制列表 (ACLs)

访问控制列表 (ACLs) 控制哪些委托人（账户成员、用户或角色）有权访问资源。 ACLs 与基于资源的策略类似，尽管它们不使用 JSON 策略文档格式。

Amazon S3 和 Amazon VPC 就是支持的服务示例 ACLs。 AWS WAF要了解更多信息 ACLs，请参阅《亚马逊简单存储服务开发者指南》中的访问控制列表 (ACL) 概述。

其他策略类型

AWS 支持其他不太常见的策略类型。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 (SCPs)- SCPs 是指定组织或组织单位 (OU) 的最大权限的 JSON 策略 AWS Organizations。 AWS Organizations 是一项用于对您的企业拥有的多 AWS 账户 项进行分组和集中管理的服务。如果您启用组织中的所有功能，则可以将服务控制策略 (SCPs) 应用于您的任何或所有帐户。SCP 限制成员账户中的实体（包括每个 AWS 账户根用户实体）的权限。有关 Organization SCPs s 和的更多信息，请参阅《AWS Organizations 用户指南》中的服务控制策略。
- 资源控制策略 (RCPs) — RCPs 是 JSON 策略，您可以使用它来设置账户中资源的最大可用权限，而无需更新附加到您拥有的每个资源的 IAM 策略。RCP 限制成员账户中资源的权限，并可能影响身份（包括身份）的有效权限 AWS 账户根用户，无论这些身份是否属于您的组织。有关 Organizations 的更多信息 RCPs，包括 AWS 服务 该支持的列表 RCPs，请参阅AWS Organizations 用户指南中的资源控制策略 (RCPs)。
- 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。要了解在涉及多种策略类型时如何 AWS 确定是否允许请求，请参阅 IAM 用户指南中的策略评估逻辑。

如何 AWS X-Ray 与 IAM 配合使用

在使用 IAM 管理对 X-Ray 的访问之前，您应了解哪些 IAM 功能可与 X-Ray 结合使用。要全面了解 X-Ray 和其他人如何 AWS 服务 使用 IAM AWS 服务 ，请参阅 IAM 用户指南中的与 I AM 配合使用。

您可以使用 AWS Identity and Access Management (IAM) 向账户中的用户和计算资源授予 X-Ray 权限。无论您的用户使用哪个客户端（控制台、 AWS SDK AWS CLI），IAM 都会在 API 级别控制对 X-Ray 服务的访问权限，以统一强制执行权限。

要使用 X-Ray 控制台查看跟踪地图和分段，只需具有读取权限即可。要启用控制台访问，请向您的 IAM 用户添加 AWSXrayReadOnlyAccess 托管策略。

要进行本地开发和测试，请创建一个具有读取和写入权限的 IAM 角色。担任该角色并存储该角色的临时凭证。您可以将这些凭据与 X-Ray 守护程序 AWS CLI、和 AWS SDK 一起使用。请参阅将 AWS CLI与临时安全凭证一起使用，了解更多信息。

要将您的检测应用程序部署到 AWS，请创建一个具有写入权限的 IAM 角色并将其分配给运行您的应用程序的资源。 AWSXRayDaemonWriteAccess包括上传跟踪的权限，以及一些支持使用采样规则的读取权限。

读写策略不包含配置加密密钥设置和采样规则的权限。用于AWSXrayFullAccess访问这些设置，或在自定义策略 APIs中添加配置。要使用您创建的客户托管密钥进行加密和解密，您还需要使用该密钥的权限。

主题

- X-Ray 基于身份的策略
- X-Ray 基于资源的策略
- 基于 X-Ray 标签的授权
- 本地运行您的应用程序
- 在中运行您的应用程序 AWS
- 用户加密权限

X-Ray 基于身份的策略

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。X-Ray 支持特定的操作、资源和条件键。要了解在 JSON 策略中使用的所有元素，请参阅《IAM 用户指南》 中的 IAM JSON 策略元素参考。

操作

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。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。

X-Ray 中的策略操作在操作前使用以下前缀：xray:。例如，要授予某人使用 X-Ray GetGroup API 操作标签组的权限，您应将xray:GetGroup操作纳入其策略中。策略语句必须包含 Action 或 NotAction 元素。X-Ray 定义了一组自己的操作，以描述您可以使用该服务执行的任务。

要在单个语句中指定多项操作，请使用逗号将它们隔开，如下所示：

```
"Action": [
      "xray:action1",
      "xray:action2"
```

您也可以使用通配符（*）指定多个操作。例如，要指定以单词 Get 开头的所有操作，包括以下操作：

```
"Action": "xray:Get*"
```

要查看 X-Ray 操作列表，请参阅 IAM 用户指南中的AWS X-Ray定义的操作。

资源

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。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": "*"
```

您可以使用 IAM 策略控制对资源的访问。对于支持资源级权限的操作，您可以使用 Amazon 资源名称 (ARN) 标识策略适用的资源。

可以在 IAM 策略中使用所有 X-Ray 操作以授予或拒绝用户使用该操作的权限。但是，并非所有 X-Ray 操作都支持资源级权限（这使您能够指定可对其执行操作的资源）。

对于不支持资源级权限的操作，您必须将“*”作为资源。

以下 X-Ray 操作支持资源级权限：

- CreateGroup
- GetGroup
- UpdateGroup
- DeleteGroup
- CreateSamplingRule
- UpdateSamplingRule
- DeleteSamplingRule

下面是 CreateGroup 操作基于身份的权限策略的示例：此示例介绍了如何使用与包含唯一 ID 作为通配符的组名称 local-users 相关的 ARN。该唯一 ID 在组创建时生成，因此策略中无法前提预测。使用 GetGroup、UpdateGroup 或 DeleteGroup 时，可将此定义为通配符或确切的 ARN（包括 ID）。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "xray:CreateGroup"
            ],
            "Resource": [
                "arn:aws:xray:eu-west-1:123456789012:group/local-users/*"
            ]
        }
    ]
}
```

下面是 CreateSamplingRule 操作基于身份的权限策略的示例：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "xray:CreateSamplingRule"
            ],
            "Resource": [
```

```
                "arn:aws:xray:eu-west-1:123456789012:sampling-rule/base-scorekeep"
            ]
        }
    ]
}
```

> **Note**
>
> 采样规则的 ARN 由其名称定义。与组不同 ARNs，采样规则没有唯一生成的 ID。

要查看 X-Ray 资源类型及其列表 ARNs，请参阅 IAM 用户指南 AWS X-Ray中的定义资源。要了解您可以在哪些操作中指定每个资源的 ARN，请参阅 AWS X-Ray定义的操作。

条件键

X-Ray 不提供任何特定于服务的条件键，但支持使用某些全局条件键。要查看所有 AWS 全局条件键，请参阅 IAM 用户指南中的AWS 全局条件上下文密钥。

示例

要查看 X-Ray 基于身份的策略的示例，请参阅 AWS X-Ray 基于身份的策略示例。

X-Ray 基于资源的策略

X-Ray 支持基于资源的策略，用于当前和未来的 AWS 服务 集成，例如 Amazon SNS 主动跟踪。基于 X-Ray 资源的策略可以由其他 AWS Management Console人更新，也可以通过 AWS SDK 或 CLI 进行更新。例如，Amazon SNS 控制台会尝试自动配置基于资源的策略，将跟踪发送给 X-Ray。以下策略文档提供手动配置 X-Ray 基于资源的策略的示例。

Example Amazon SNS 主动跟踪的 X-Ray 基于资源的策略示例

以下示例策略文档指定了 Amazon SNS 将跟踪数据发送给 X-Ray 所需要的权限：

```
{
    Version: "2012-10-17",
    Statement: [
      {
        Sid: "SNSAccess",
        Effect: Allow,
        Principal: {
```

```
          Service: "sns.amazonaws.com",
        },
        Action: [
          "xray:PutTraceSegments",
          "xray:GetSamplingRules",
          "xray:GetSamplingTargets"
        ],
        Resource: "*",
        Condition: {
          StringEquals: {
            "aws:SourceAccount": "account-id"
          },
          StringLike: {
            "aws:SourceArn": "arn:partition:sns:region:account-id:topic-name"
          }
        }
      }
    ]
  }
```

使用 CLI 创建基于资源的策略，赋予 Amazon SNS 将跟踪数据发送给 X-Ray 的权限：

```
aws xray put-resource-policy --policy-name MyResourcePolicy --policy-document
 '{ "Version": "2012-10-17", "Statement": [ { "Sid": "SNSAccess", "Effect": "Allow",
 "Principal": { "Service": "sns.amazonaws.com" }, "Action": [ "xray:PutTraceSegments",
 "xray:GetSamplingRules", "xray:GetSamplingTargets" ], "Resource": "*",
 "Condition": { "StringEquals": { "aws:SourceAccount": "account-id" }, "StringLike":
 { "aws:SourceArn": "arn:partition:sns:region:account-id:topic-name" } } } ] }'
```

要使用这些示例，请将*partition*、*regionaccount-id*、和*topic-name*替换为您的特定 AWS 分区、区域、账户 ID 和 Amazon SNS 主题名称。如需赋予所有 Amazon SNS 主题将跟踪数据发送给 X-Ray 的权限，请将主题名称替换为 *。

基于 X-Ray 标签的授权

您可以将标签附加到 X-Ray 组或采样规则，或在发给 X-Ray 请求中传递标签。要基于标签控制访问，您需要使用 `xray: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。有关标记 X-Ray 资源的更多信息，请参阅标记 X-Ray 采样规则和组。

要查看基于身份的策略（用于根据资源上的标签来限制对该资源的访问）的示例，请参阅根据标签管理对 X-Ray 组和采样规则的访问权限。

本地运行您的应用程序

您的已检测应用程序将跟踪数据发送到 X-Ray 进程守护程序。进程守护程序缓存分段文档，并分批将它们上传到 X-Ray 服务。进程守护程序需要写入权限以将跟踪数据和遥测数据上传到 X-Ray 服务。

当在本地运行进程守护程序时，创建一个 IAM 角色，担任该角色并将临时凭证存储在环境变量中，或用户文件夹中名为 `.aws` 的文件夹中名为 `credentials` 的文件中。请参阅将 AWS CLI与临时安全凭证一起使用，了解更多信息。

Example ~/.aws/credentials

```
[default]
aws_access_key_id={access key ID}
aws_secret_access_key={access key}
aws_session_token={AWS session token}
```

如果您已经配置了用于 AWS SDK 或的凭证 AWS CLI，则守护程序可以使用这些凭据。如果有多个配置文件可用，则该进程守护程序使用默认配置文件。

在中运行您的应用程序 AWS

当您在上运行应用程序时 AWS，使用角色向运行守护程序的 Amazon EC2 实例或 Lambda 函数授予权限。

- Amazon Elastic Compute Cloud (Amazon EC2) — 创建 IAM 角色并将其作为 EC2 实例配置文件附加到实例。
- Amazon Elastic Container Service (Amazon ECS) - 创建一个 IAM 角色并将其作为容器实例 IAM 角色附加到容器实例。
- AWS Elastic Beanstalk （ Elastic Beanstalk) — Elastic Beanstalk 在其默认实例配置文件中包含 X-Ray 权限。您可以使用该默认实例配置文件，或在自定义实例配置文件中添加写入权限。
- AWS Lambda (Lambda)-向函数的执行角色添加写入权限。

如何创建角色来使用 X-Ray

1. 打开 IAM 管理控制台。
2. 选择角色。
3. 选择创建新角色。

4. 对于角色名称，键入 **xray-application**。选择下一步。
5. 对于角色类型，请选择 Amazon EC2。
6. 附加以下托管策略以向您应用程序授予对 AWS 服务的访问权限。

   - AWSXRayDaemonWriteAccess— 授予 X-Ray 守护程序上传跟踪数据的权限。

   如果您的应用程序使用 AWS SDK 访问其他服务，请添加授予对这些服务的访问权限的策略。
7. 选择下一步。
8. 请选择创建角色。

用户加密权限

默认情况下，X-Ray 将加密所有跟踪数据，您可以将它配置为使用您管理的密钥。如果您选择 AWS Key Management Service 客户管理的密钥，则需要确保该密钥的访问策略允许您向 X-Ray 授予使用它进行加密的权限。您账户中的其他用户还需要访问该密钥来查看在 X-Ray 控制台中加密的跟踪数据。

对于客户托管密钥，请使用允许以下操作的访问策略配置您的密钥：

- 在 X-Ray 中配置密钥的用户有权调用 `kms:CreateGrant` 和 `kms:DescribeKey`。
- 可以访问加密跟踪数据的用户有权调用 `kms:Decrypt`。

当您在 IAM 控制台的密钥配置部分中为密钥用户组添加一名用户时，他们同时拥有这两项操作的权限。只需在密钥策略上设置权限，因此您不需要对用户、群组或角色 AWS KMS 拥有任何权限。有关更多信息，请参阅《 AWS KMS 开发人员指南》中的使用密钥策略。

对于默认加密，或者如果您选择 AWS 托管 CMK (`aws/xray`)，则权限取决于谁有权访问 X-R APIs ay。包含在 `AWSXrayFullAccess` 中的具有对 `PutEncryptionConfig` 的访问权限的任何人都可以更改加密配置。要防止用户更改加密密钥，请勿向这些用户授予使用 `PutEncryptionConfig` 的权限。

AWS X-Ray 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 X-Ray 资源的权限。他们也无法使用 AWS Management Console AWS CLI、或 AWS API 执行任务。管理员必须创建 IAM policy，以便为用户和角色授予权限以对所需的指定资源执行特定的 API 操作。然后，管理员必须将这些策略附加到需要这些权限的用户或组。

要了解如何使用这些示例 JSON 策略文档创建 IAM 基于身份的策略，请参阅《IAM 用户指南》中的在 JSON 选项卡上创建策略。

主题

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 X-Ray 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：

- 开始使用 AWS 托管策略并转向最低权限权限 — 要开始向用户和工作负载授予权限，请使用为许多常见用例授予权限的AWS 托管策略。它们在你的版本中可用 AWS 账户。我们建议您通过定义针对您的用例的 AWS 客户托管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 托管式策略或工作职能的AWS 托管式策略。
- 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。如果服务操作是通过特定的方式使用的，则也可以使用条件来授予对服务操作的访问权限 AWS 服务，例如 AWS CloudFormation。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》中的使用 IAM Access Analyzer 验证策略。
- 需要多重身份验证 (MFA)-如果 AWS 账户您的场景需要 IAM 用户或根用户，请启用 MFA 以提高安全性。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》中的使用 MFA 保护 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 中的安全最佳实践。

使用 X-Ray 控制台

要访问 AWS X-Ray 控制台，您必须拥有一组最低权限。这些权限必须允许您列出和查看有关您的 X-Ray 资源的详细信息 AWS 账户。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。

为确保这些实体仍然可以使用 X-Ray 控制台，请将AWSXRayReadOnlyAccess AWS 托管策略附加到这些实体。X-Ray 的 IAM 托管策略中更加详细地介绍了此策略。有关更多信息，请参阅《IAM 用户指南》中的为用户添加权限。

对于仅调用 AWS CLI 或 AWS API 的用户，您无需为其设置最低控制台权限。相反，只允许访问与您尝试执行的 API 操作相匹配的操作。

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式策略。此策略包括在控制台上或使用 AWS CLI 或 AWS API 以编程方式完成此操作的权限。

```
{
    "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": "*"
        }
    ]
}
```

根据标签管理对 X-Ray 组和采样规则的访问权限

您可以在基于身份的策略中使用条件，以便基于标签控制对 X-Ray 组和采样规则的访问。以下示例策略可用于拒绝用户角色创建、删除或更新具有标签 `stage:prod` 或 `stage:preprod` 的组的权限。有关标记 X-Ray 采样规则和组的更多信息，请参阅 [标记 X-Ray 采样规则和组]。

如需拒绝用户对创建、更新或删除具有标签 `stage:prod` 或 `stage:preprod` 的组的访问权限，请为该用户分配具有类似于以下策略的角色。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAllXRay",
            "Effect": "Allow",
            "Action": "xray:*",
            "Resource": "*"
        },
        {
            "Sid": "DenyCreateGroupWithStage",
            "Effect": "Deny",
            "Action": [
                "xray:CreateGroup"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/stage": [
                        "preprod",
                        "prod"
                    ]
                }
```

```
            }
        },
        {
            "Sid": "DenyUpdateGroupWithStage",
            "Effect": "Deny",
            "Action": [
                "xray:UpdateGroup",
                "xray:DeleteGroup"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/stage": [
                        "preprod",
                        "prod"
                    ]
                }
            }
        }
    ]
}
```

如需拒绝创建采样规则，请使用 aws:RequestTag 指明标签不能作为创建请求的一部分进行传递。如需拒绝更新或删除采样规则，请使用 aws:ResourceTag 拒绝基于这些资源应用的标签的操作。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAllXRay",
            "Effect": "Allow",
            "Action": "xray:*",
            "Resource": "*"
        },
        {
            "Sid": "DenyCreateSamplingRuleWithStage",
            "Effect": "Deny",
            "Action": "xray:CreateSamplingRule",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/stage": [
                        "preprod",
```

```
                        "prod"
                    ]
                }
            }
        },
        {
            "Sid": "DenyUpdateSamplingRuleWithStage",
            "Effect": "Deny",
            "Action": [
                "xray:UpdateSamplingRule",
                "xray:DeleteSamplingRule"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/stage": [
                        "preprod",
                        "prod"
                    ]
                }
            }
        }
    ]
}
```

可以将这些策略（或将它们整合为单一策略，然后再附加策略）附加到账户中的用户。如果用户想要对组或采样规则进行更改，则相应的组或采样规则不得标记为 `stage=prepod` 或 `stage=prod`。条件标签键 `Stage` 匹配 `Stage` 和 `stage`，因为条件键名称不区分大小写。有关条件块的更多信息，请参阅《IAM 用户指南》中的 IAM JSON 策略元素：条件。

角色中附加有以下策略的用户无法将标签 `role:admin` 添加到资源，且无法从具有关联的 `role:admin` 的资源中删除标签。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAllXRay",
            "Effect": "Allow",
            "Action": "xray:*",
            "Resource": "*"
        },
        {
```

```
            "Sid": "DenyRequestTagAdmin",
            "Effect": "Deny",
            "Action": "xray:TagResource",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/role": "admin"
                }
            }
        },
        {
            "Sid": "DenyResourceTagAdmin",
            "Effect": "Deny",
            "Action": "xray:UntagResource",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/role": "admin"
                }
            }
        }
    ]
}
```

X-Ray 的 IAM 托管策略

为方便授权，IAM 支持将托管策略用于每个服务。服务可以在发布新权限时使用新权限更新这些托管策略 APIs。 AWS X-Ray 为只读、只写和管理员用例提供托管策略。

- `AWSXrayReadOnlyAccess`— 读取使用 X-Ray 控制台或 AWS SDK 从 X-Ray API 获取跟踪数据、轨迹地图、见解和 X-Ray 配置的权限。 AWS CLI包括可观察性访问管理器 (OAM) `oam:ListSinks` 和`oam:ListAttachedSinks`权限，允许控制台查看作为CloudWatch 跨账户可观察性一部分的源账户共享的跟踪记录。`BatchGetTraceSummaryById`和 `GetDistinctTraceGraphs` API 操作不打算由您的代码调用，也不包含在 AWS CLI 和中 AWS SDKs。

  ```
  {
      "Version": "2012-10-17",
      "Statement": [
          {
              "Effect": "Allow",
              "Action": [
  ```

```
                "xray:GetSamplingRules",
                "xray:GetSamplingTargets",
                "xray:GetSamplingStatisticSummaries",
                "xray:BatchGetTraces",
                "xray:BatchGetTraceSummaryById",
                "xray:GetDistinctTraceGraphs",
                "xray:GetServiceGraph",
                "xray:GetTraceGraph",
                "xray:GetTraceSummaries",
                "xray:GetGroups",
                "xray:GetGroup",
                "xray:ListTagsForResource",
                "xray:ListResourcePolicies",
                "xray:GetTimeSeriesServiceStatistics",
                "xray:GetInsightSummaries",
                "xray:GetInsight",
                "xray:GetInsightEvents",
                "xray:GetInsightImpactGraph",
                "oam:ListSinks"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "oam:ListAttachedLinks"
            ],
            "Resource": "arn:aws:oam:*:*:sink/*"
        }

}
```

- `AWSXRayDaemonWriteAccess`— 写入使用 X-Ray 守护程序或 AWS SDK 将分段文档和遥测数据上传到 X-Ray API 的权限。 AWS CLI包含读取权限以获取采样规则并报告采样结果。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
```

```
                "xray:PutTraceSegments",
                "xray:PutTelemetryRecords",
                "xray:GetSamplingRules",
                "xray:GetSamplingTargets",
                "xray:GetSamplingStatisticSummaries"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

- `AWSXrayCrossAccountSharingConfiguration` - 授予创建、管理和查看 Observability Access Manager 链接的权限，在账户之间实现 X-Ray 资源共享。用于启用来源账户和监控[CloudWatch 账户之间的跨账户可观察性](#)。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "xray: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/*"
            ]
        }
    ]

}
```

- `AWSXrayFullAccess`— 使用所有 X-Ray 的权限 APIs，包括读取权限、写入权限以及配置加密密钥设置和采样规则的权限。包括可观察性访问管理器 (OAM) `oam:ListSinks` 和`oam:ListAttachedSinks`权限，允许控制台查看作为CloudWatch 跨账户可观察性一部分的源账户共享的跟踪记录。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "xray:*",
                "oam:ListSinks"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "oam:ListAttachedLinks"
            ],
            "Resource": "arn:aws:oam:*:*:sink/*"
        }
    ]
}
```

将托管策略添加到 IAM 用户、组或角色

1. 打开 IAM 管理控制台。
2. 打开与您的实例配置文件、IAM 用户或 IAM 组关联的角色。

3. 在权限下，附加托管策略。

AWS 托管策略的 X-Ray 更新

查看自该服务开始跟踪这些更改以来，X-Ray AWS 托管策略更新的详细信息。有关此页面更改的自动提示，请订阅 X-Ray 文档历史记录页面上的 RSS 源。

更改	描述	日期
X-Ray 的 IAM 托管策略 - 添加了新的 `AWSXrayCrossAccountSharingConfiguration` ，并更新了 `AWSXrayReadOnlyAccess` 和 `AWSXrayFullAccess` 策略。	X-Ray 在这些策略中添加了可观察性访问管理器 (OAM) 权限`oam:ListSinks` ，允许控制台查看作为CloudWatch 跨账户可观察性一部分的源账户共享的跟踪。`oam:ListAttachedSinks`	2022 年 11 月 27 日
X-Ray 的 IAM 托管策略 - 更新了 `AWSXrayReadOnlyAccess` 策略。	X-Ray 添加了一项 API 操作，`ListResourcePolicies` 。	2022 年 11 月 15 日
使用 X-Ray 控制台 - 更新了 `AWSXrayReadOnlyAccess` 策略	X-Ray 添加了两项 API 操作，`BatchGetTraceSummaryById` 和 `GetDistinctTraceGraphs` 。 这些操作不应由您的代码调用。因此，这些 API 操作不包含在 AWS CLI 和中 AWS SDKs。	2022 年 11 月 11 日

在 IAM 策略中指定资源

您可以使用 IAM 策略控制对资源的访问。对于支持资源级权限的操作，您可以使用 Amazon 资源名称 (ARN) 标识策略适用的资源。

可以在 IAM 策略中使用所有 X-Ray 操作以授予或拒绝用户使用该操作的权限。但是，并非所有 X-Ray 操作都支持资源级权限（这使您能够指定可对其执行操作的资源）。

对于不支持资源级权限的操作，您必须将“*”作为资源。

以下 X-Ray 操作支持资源级权限：

- CreateGroup
- GetGroup
- UpdateGroup
- DeleteGroup
- CreateSamplingRule
- UpdateSamplingRule
- DeleteSamplingRule

下面是 CreateGroup 操作基于身份的权限策略的示例：此示例介绍了如何使用与包含唯一 ID 作为通配符的组名称 local-users 相关的 ARN。该唯一 ID 在组创建时生成，因此策略中无法前提预测。使用 GetGroup、UpdateGroup 或 DeleteGroup 时，可将此定义为通配符或确切的 ARN（包括 ID）。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "xray:CreateGroup"
            ],
            "Resource": [
                "arn:aws:xray:eu-west-1:123456789012:group/local-users/*"
            ]
        }
    ]
}
```

下面是 CreateSamplingRule 操作基于身份的权限策略的示例：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Action": [
                "xray:CreateSamplingRule"
            ],
            "Resource": [
                "arn:aws:xray:eu-west-1:123456789012:sampling-rule/base-scorekeep"
            ]
        }
    ]
}
```

> **Note**
>
> 采样规则的 ARN 由其名称定义。与组不同 ARNs，采样规则没有唯一生成的 ID。

对 AWS X-Ray 身份和访问进行故障排除

使用以下信息可帮助您诊断和修复在使用 X-Ray 和 IAM 时可能遇到的常见问题。

主题

- 我无权在 X-Ray 中执行操作
- 我无权执行 iam：PassRole
- 我是管理员并希望允许其他人访问 X-Ray
- 我想允许我以外的人访问我 AWS 账户 的 X-Ray 资源

我无权在 X-Ray 中执行操作

如果 AWS Management Console 告诉您您无权执行某项操作，则必须联系管理员寻求帮助。管理员是向您提供登录凭证的人。

当 `mateojackson` 用户尝试使用控制台查看有关采样规则的详细信息，但不具有 `xray:GetSamplingRules` 权限时，会发生以下示例错误。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to
 perform: xray:GetSamplingRules on resource: arn:${Partition}:xray:${Region}:
${Account}:sampling-rule/${SamplingRuleName}
```

在这种情况下，Mateo 请求管理员更新其策略，以允许他使用 `xray:GetSamplingRules` 操作访问采样规则资源。

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，则必须更新策略以允许您将角色传递给 X-Ray。

有些 AWS 服务 允许您将现有角色传递给该服务，而不是创建新的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 X-Ray 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，请联系您的 AWS 管理员。您的管理员是提供登录凭证的人。

我是管理员并希望允许其他人访问 X-Ray

要允许其他人访问 X-Ray，您必须向需要访问权限的人员或应用程序授予权限。如果使用 AWS IAM Identity Center 管理人员和应用程序，则可以向用户或组分配权限集来定义其访问权限级别。权限集会自动创建 IAM 策略并将其分配给与人员或应用程序关联的 IAM 角色。有关更多信息，请参阅《AWS IAM Identity Center 用户指南》中的权限集。

如果未使用 IAM Identity Center，则必须为需要访问的人员或应用程序创建 IAM 实体（用户或角色）。然后，您必须将策略附加到实体，以便在 X-Ray 中向其授予正确的权限。授予权限后，向用户或应用程序开发人员提供凭证。他们将使用这些凭证访问 AWS。要了解有关创建 IAM 用户、组、策略和权限的更多信息，请参阅《IAM 用户指南》中的 IAM 身份和 IAM 中的策略和权限。

我想允许我以外的人访问我 AWS 账户 的 X-Ray 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，可以代入角色。对于支持基于资源的策略或访问控制列表 (ACLs) 的服务，您可以使用这些策略向人们授予访问您的资源的权限。

要了解更多信息，请参阅以下内容：

- 要了解 X-Ray 是否支持这些特征，请参阅 如何 AWS X-Ray 与 IAM 配合使用。
- 要了解如何提供对您拥有的资源的访问权限 AWS 账户 ，请参阅 IAM 用户指南中的向您拥有 AWS 账户 的另一个 IAM 用户提供访问权限。

- 要了解如何向第三方提供对您的资源的访问权限 AWS 账户，请参阅 IAM 用户指南中的向第三方提供访问权限。 AWS 账户
- 要了解如何通过身份联合验证提供访问权限，请参阅《IAM 用户指南》中的为经过外部身份验证的用户（身份联合验证）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。

登录和监控 AWS X-Ray

监控是保持您的 AWS 解决方案的可靠性、可用性和性能的重要方面。您应该从 AWS 解决方案的所有部分收集监控数据，以便在出现多点故障时可以更轻松地对其进行调试。 AWS 提供了多种用于监控您的 X-Ray 资源和响应潜在事件的工具：

AWS CloudTrail 日志

AWS X-Ray 与集成 AWS CloudTrail 以记录用户、角色或 AWS 服务在 X-Ray 中执行的 API 操作。您可以使用 CloudTrail 实时监控 X-Ray API 请求，并将日志存储在 Amazon S3、Amazon L CloudWatch ogs 和 Amazon Ev CloudWatch ents 中。有关更多信息，请参阅 使用记录 X-Ray API 调用 AWS CloudTrail。

AWS Config 追踪

AWS X-Ray 与集成 AWS Config 以记录对 X-Ray 加密资源所做的配置更改。您可以使用 AWS Config 清点 X-Ray 加密资源、审计 X-Ray 配置历史记录以及根据资源更改发送通知。有关更多信息，请参阅 使用跟踪 X-Ray 加密配置的更改 AWS Config。

亚马逊 CloudWatch 监控

您可以使用适用于 Java 的 X-Ray SDK 从您收集的 X-Ray 区段中发布未采样的亚马逊 CloudWatch 指标。这些指标来自分段的开始和结束时间以及错误、故障和限制状态标志。使用这些指标可暴露子分段里的重试和依赖项问题。有关更多信息，请参阅 AWS X-Ray 适用于 Java 的 X-Ray SDK 的指标。

合规性验证 AWS X-Ray

要了解是否属于特定合规计划的范围，请参阅AWS 服务 “按合规计划划分的范围” ”，然后选择您感兴趣的合规计划。 AWS 服务 有关一般信息，请参阅AWS 合规计划AWS。

您可以使用下载第三方审计报告 AWS Artifact。有关更多信息，请参阅中的 “下载报告” 中的 “ AWS Artifact。

您在使用 AWS 服务 时的合规责任取决于您的数据的敏感性、贵公司的合规目标以及适用的法律和法规。 AWS 提供了以下资源来帮助实现合规性：

- Security Compliance & Governance：这些解决方案实施指南讨论了架构考虑因素，并提供了部署安全性和合规性功能的步骤。
- 符合 HIPAA 要求的服务参考：列出符合 HIPAA 要求的服务。并非所有 AWS 服务 人都符合 HIPAA 资格。
- AWS 合规资源AWS — 此工作簿和指南集可能适用于您所在的行业和所在地区。
- AWS 客户合规指南 — 从合规角度了解责任共担模式。这些指南总结了保护的最佳实践， AWS 服务并将指南映射到跨多个框架（包括美国国家标准与技术研究院 (NIST)、支付卡行业安全标准委员会 (PCI) 和国际标准化组织 (ISO)）的安全控制。
- 使用AWS Config 开发人员指南中的规则评估资源 — 该 AWS Config 服务评估您的资源配置在多大程度上符合内部实践、行业准则和法规。
- AWS Security Hub— 这 AWS 服务 提供了您内部安全状态的全面视图 AWS。Security Hub 通过安全控制措施评估您的 AWS 资源并检查其是否符合安全行业标准和最佳实践。有关受支持服务及控制措施的列表，请参阅 Security Hub 控制措施参考。
- Amazon GuardDuty — 它通过监控您的 AWS 账户环境中是否存在可疑和恶意活动，来 AWS 服务检测您的工作负载、容器和数据面临的潜在威胁。 GuardDuty 通过满足某些合规性框架规定的入侵检测要求，可以帮助您满足各种合规性要求，例如 PCI DSS。
- AWS Audit Manager— 这 AWS 服务 可以帮助您持续审计 AWS 使用情况，从而简化风险管理以及对法规和行业标准的合规性。

韧性在 AWS X-Ray

AWS 全球基础设施是围绕 AWS 区域 可用区构建的。 AWS 区域 提供多个物理隔离和隔离的可用区，这些可用区通过低延迟、高吞吐量和高度冗余的网络连接。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失效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架构相比，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。

中的基础设施安全 AWS X-Ray

作为一项托管服务 AWS X-Ray ，受 AWS 全球网络安全的保护。有关 AWS 安全服务以及如何 AWS 保护基础设施的信息，请参阅AWS 云安全。要使用基础设施安全的最佳实践来设计您的 AWS 环境，请参阅 S AWS ecurity Pillar Well-Architected Fram ework 中的基础设施保护。

您可以使用 AWS 已发布的 API 调用通过网络访问 X-Ray。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。

AWS X-Ray 与 VPC 终端节点一起使用

如果您使用亚马逊虚拟私有云（亚马逊 VPC）托管 AWS 资源，则可以在您的 VPC 和 X-Ray 之间建立私有连接。这让 Amazon VPC 中的资源能够与 X-Ray 服务进行通信而不用访问公共互联网。

Amazon VPC 可用于在您定义的虚拟网络中启动 AWS 资源。 AWS 服务 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 VPC 连接到 X-Ray，请定义一个接口 VPC 端点。该端点提供了到 X-Ray 的可靠、可扩展的连接，无需 Internet 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。

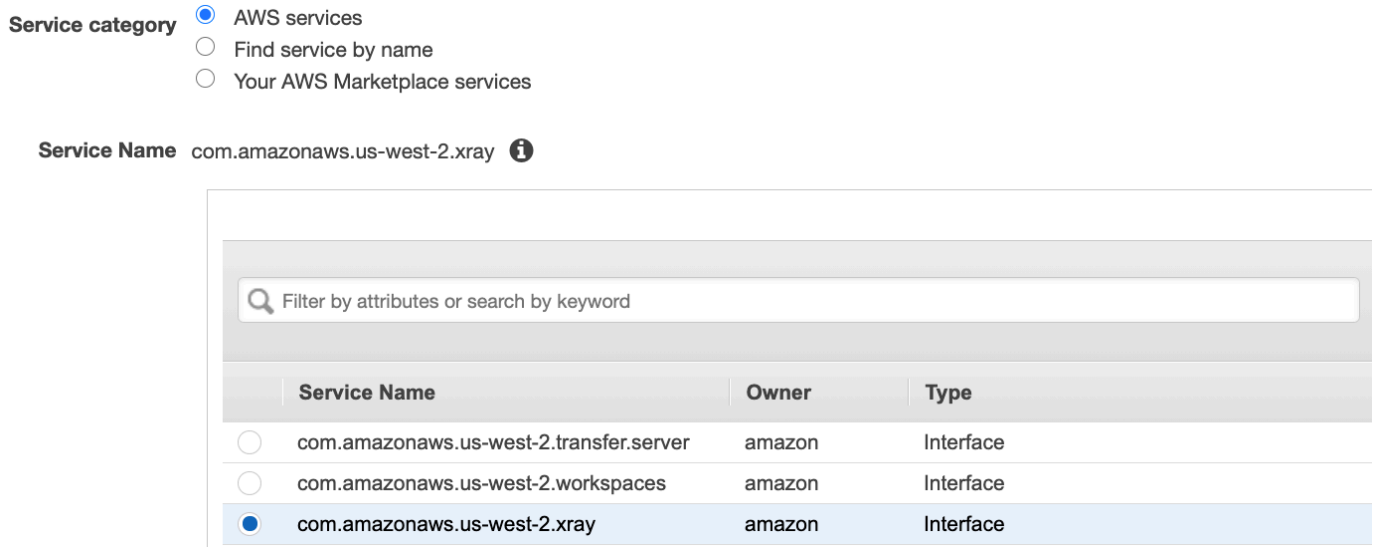
接口 VPC 终端节点由 AWS PrivateLink一种 AWS 技术提供支持，该技术 AWS 服务 通过使用带有私有 IP 地址的弹性网络接口实现两者之间的私密通信。有关更多信息，请参阅《Amazon VPC 用户指南》中的 “新增内容” AWS 服务博客文章和《入门》。 AWS PrivateLink

要确保可以在所选的 X-Ray 中创建 VPC 终端节点 AWS 区域，请参阅支持的区域。

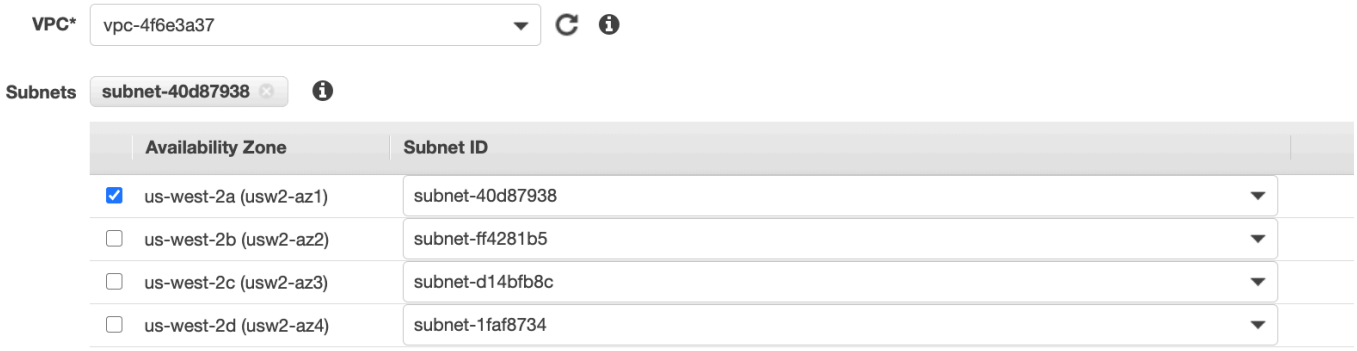
为 X-Ray 创建 VPC 端点

要开始将您的 X-Ray 与 VPC 一起使用，请为 X-Ray 创建接口 VPC 端点。

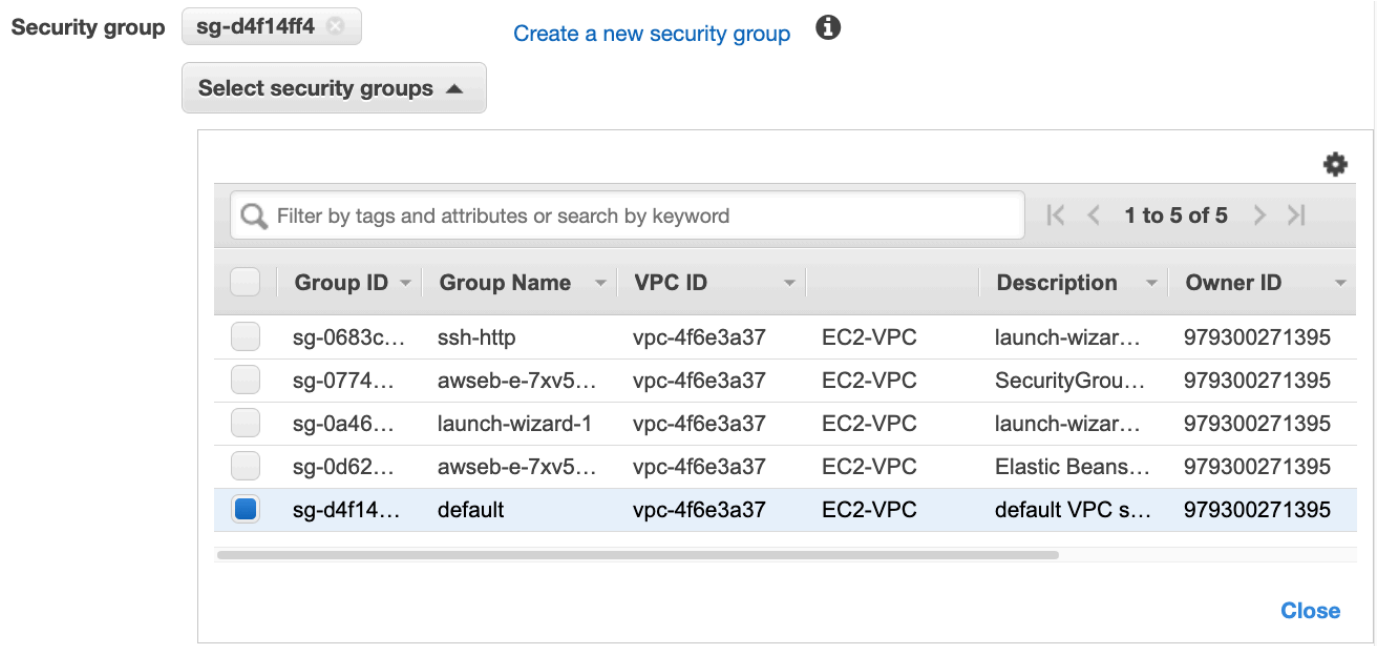
1. 打开位于 https://console.aws.amazon.com/vpc/ 的 Amazon VPC 控制台。
2. 在导航窗格中导航到端点，然后选择创建端点。
3. 搜索并选择 AWS X-Ray 服务名称:`com.amazonaws.region.xray`.



4. 选择您想要的 VPC，然后在 VPC 中选择使用接口端点的子网。所选子网中创建一个端点网络接口。您可以在不同的可用区中指定多个子网（在服务支持的情况下），以帮助确保您的接口端点能够在出现可用区故障时复原。如果执行此操作，将在您指定的每个子网中创建一个接口网络接口。



5. （可选）默认情况下，端点启用私有 DNS，以使您能够使用默认的 DNS 主机名向 X-Ray 发出请求。您可以选择将其禁用。
6. 指定要与端点网络接口关联的安全组。



7. （可选）指定自定义策略来控制对 X-Ray 服务的访问权限。默认情况下，允许完全访问。

控制对 X-Ray VPC 端点的访问

VPC 端点策略是一种 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 IAM 用户策略或服务特定的策略。这是一个单独的策略，用于控制从端点中对指定服务进行的访问。端点策略必须采用 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权限。

VPC 端点策略让您可以控制对多种 X-Ray 操作的权限。例如，您可以创建一个仅允许 PutTraceSegment 和拒绝所有其他操作的策略。这会限制 VPC 中的工作负载和服务仅将跟踪数据发送给 X-Ray，并拒绝检索数据、更改加密配置或创建/更新组等任何其他操作。

下面是用于 X-Ray 的端点策略示例。该策略允许通过 VPC 连接到 X-Ray 的用户将分段数据发送到 X-Ray，还禁止他们执行其他 X-Ray 操作。

```
{"Statement": [
    {"Sid": "Allow PutTraceSegments",
      "Principal": "*",
      "Action": [
        "xray:PutTraceSegments"
      ],
      "Effect": "Allow",
```

```
      "Resource": "*"
    }
  ]
}
```

编辑 X-Ray 的 VPC 端点策略

1. 打开位于 https://console.aws.amazon.com/vpc/ 的 Amazon VPC 控制台。
2. 在导航窗格中，选择端点。
3. 如果您尚未为 X-Ray 创建端点，请按照为 X-Ray 创建 VPC 端点中的步骤操作。
4. 选择 com.amazonaws。***region***.xray 端点，然后选择 “策略” 选项卡。
5. 选择编辑策略，然后进行更改。

支持的区域

X-Ray 目前支持以下方面的 VPC 终端节点 AWS 区域：

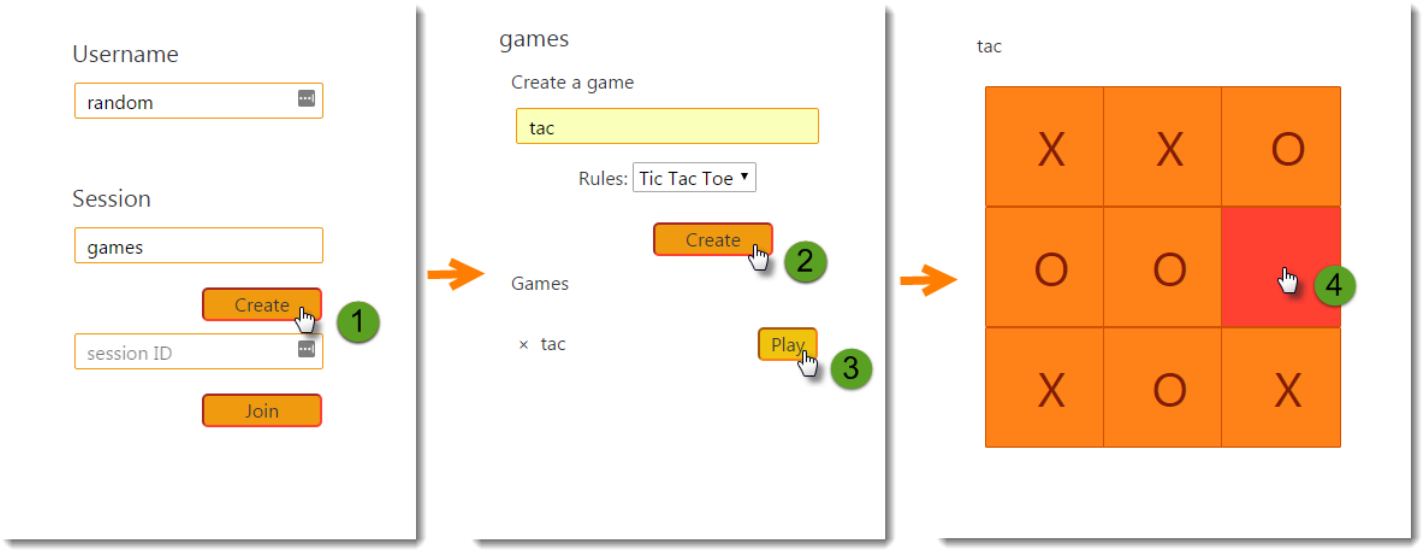
- 美国东部（俄亥俄州）
- 美国东部（弗吉尼亚州北部）
- 美国西部（加利福尼亚北部）
- 美国西部（俄勒冈州）
- 非洲（开普敦）
- 亚太地区（香港）
- Asia Pacific (Mumbai)
- 亚太地区（大阪）
- 亚太地区（首尔）
- 亚太地区（新加坡）
- 亚太地区（悉尼）
- 亚太地区（东京）
- 加拿大（中部）
- 欧洲地区（法兰克福）
- 欧洲地区（爱尔兰）
- 欧洲地区（伦敦）

- 欧洲地区（米兰）
- 欧洲地区（巴黎）
- 欧洲地区（斯德哥尔摩）
- 中东（巴林）
- 南美洲（圣保罗）
- AWS GovCloud （美国东部）
- AWS GovCloud （美国西部）

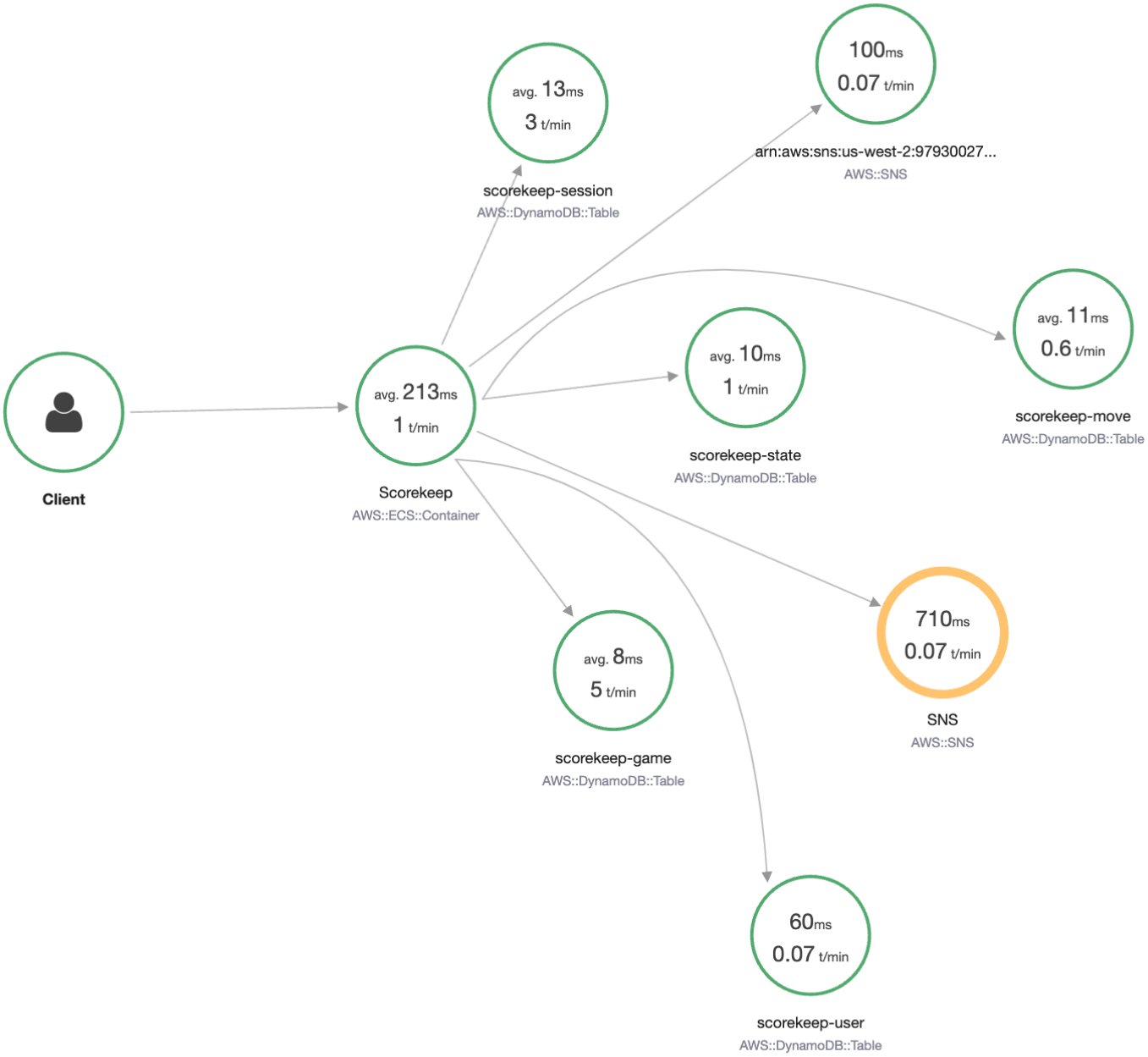
AWS X-Ray 示例应用程序

上 GitHub提供的 AWS X-Ray eb-java-scorekeep示例应用程序展示了如何使用 AWS X-Ray SDK 来检测传入的 HTTP 调用、DynamoDB SDK 客户端和 HTTP 客户端。该示例应用程序用于 AWS CloudFormation 创建 DynamoDB 表、在实例上编译 Java 代码以及运行 X-Ray 守护程序，无需任何其他配置。

请参阅 Scorekeeep 教程，使用或开始安装和使用带检测功能的 AWS Management Console 示例应用程序。 AWS CLI



该示例包括前端 Web 应用程序、所调用的 API 以及它用于存储数据的 DynamoDB 表。包含过滤器、插件和经过检测的 AWS SDK 客户端的基本插件显示在项目的`xray-gettingstarted`分支中。这是您在入门教程中部署的分支。由于此分支只包含基本分析，您可以根据 `master` 分支比较差异，以快速理解基本分析。



该应用程序示例在这些文件中显示基本检测：

- HTTP 请求筛选器 - `WebConfig.java`
- AWS SDK 客户端工具 — `build.gradle`

该应用程序的`xray`分支包括使用注解HTTPClient、SQL 查询、自定义子分段、检测AWS Lambda函数以及检测过的初始化代码和脚本。

为了支持用户登录和在浏览器中 适用于 JavaScript 的 AWS SDK 使用，该`xray-cognito`分支机构添加了 Amazon Cognito 来支持用户身份验证和授权。利用从 Amazon Cognito 检索到的凭证，Web 应用程序还会将跟踪数据发送到 X-Ray，以从客户端的视角记录请求信息。浏览器客户端在跟踪地图中显示为自己的节点，并记录其他信息，包括用户正在查看的页面的 URL 和用户的 ID。

最后，`xray-worker` 分支将添加独立运行的检测过的 Lambda 函数，并处理来自 Amazon SQS 队列的项目。每当游戏结束时，Scorekeep 就会向队列添加一个项目。由 CloudWatch 事件触发的 Lambda 工作程序每隔几分钟从队列中提取项目，然后对其进行处理，以便将游戏记录存储在 Amazon S3 中以供分析。

主题

Scorekeep 示例应用程序入门

本教程使用 Scorekeeep 示例应用程序的`xray-gettingstarted`分支，该分支用于 AWS CloudFormation 创建和配置在 Amazon ECS 上运行示例应用程序和 X-Ray 守护程序的资源。该应用程序使用 Spring 框架来实现 JSON Web API，并使用将数据保存 适用于 Java 的 AWS SDK 到 Amazon DynamoDB。应用程序中的 servlet 过滤器会检测应用程序处理的所有传入请求，而 AWS SDK 客户端上的请求处理程序则用于监测 DynamoDB 的下游调用。

您可以使用 AWS Management Console 或来学习本教程 AWS CLI。

Sections

- 使用以下命令安装 Scorekeeep 应用程序 CloudFormation
- 生成跟踪数据
- 在中查看追踪地图 AWS Management Console
- 配置 Amazon SNS 通知
- 浏览应用程序示例
- 可选：最低权限策略
- 清理
- 后续步骤

先决条件

本教程 AWS CloudFormation 用于创建和配置运行示例应用程序和 X-Ray 守护程序的资源。安装和运行本教程需要满足以下先决条件：

1. 如果您使用权限有限的 IAM 用户，请在 IAM 控制台中添加以下用户策略：
   - `AWSCloudFormationFullAccess`— 访问和使用 CloudFormation
   - `AmazonS3FullAccess`— CloudFormation 使用将模板文件上传到 AWS Management Console
   - `IAMFullAccess`— 创建 Amazon ECS 和亚马逊 EC2 实例角色
   - `AmazonEC2FullAccess`— 创建 Amazon EC2 资源
   - `AmazonDynamoDBFullAccess` - 用于创建 DynamoDB 表
   - `AmazonECS_FullAccess` - 用于创建 Amazon ECS 资源
   - `AmazonSNSFullAccess` - 用于创建 Amazon SNS 主题
   - `AWSXrayReadOnlyAccess` - 用于查看 X-Ray 控制台中跟踪地图和跟踪的权限
2. 要使用完成本教程 AWS CLI，请安装 CLI 版本 2.7.9 或更高版本，并使用上一步中的用户配置 CLI。使用用户配置时，请确保已 AWS CLI 配置区域。如果未配置区域，则需要将 `--region` *`AWS-REGION`* 附加到每一个 CLI 命令。
3. 确保已安装 Git，以便克隆示例应用程序存储库。
4. 使用以下代码示例克隆 Scorekeep 存储库的 `xray-gettingstarted` 分支：

```
git clone https://github.com/aws-samples/eb-java-scorekeep.git xray-scorekeep -b
 xray-gettingstarted
```

使用以下命令安装 Scorekeeep 应用程序 CloudFormation

AWS Management Console

使用安装示例应用程序 AWS Management Console

1. 打开 CloudFormation 控制台
2. 选择创建堆栈，然后从下列菜单中选择使用新资源。
3. 在指定模板部分，选择上传模板文件。
4. 选择选择文件，导航到克隆 git 存储库时创建的 `xray-scorekeep/cloudformation` 文件夹，然后选择 `cf-resources.yaml` 文件。
5. 选择下一步以继续。
6. 在堆栈名称文本框中输入 `scorekeep`，然后选择页面底部的下一步以继续。请注意，本教程的其余部分假设堆栈已命名为 `scorekeep`。
7. 滚动到配置堆栈选项页面底部，选择下一步以继续。
8. 滚动到 “查看” 页面底部，选中确认 CloudFormation 可以创建具有自定义名称的 IAM 资源的复选框，然后选择创建堆栈。
9. CloudFormation 堆栈正在创建中。堆栈状态将在五分钟左右保持为 `CREATE_IN_PROGRESS`，然后变为 `CREATE_COMPLETE`。状态将会定期更新，也可以刷新页面。

AWS CLI

使用安装示例应用程序 AWS CLI

1. 导航到在教程更早时候克隆的 `cloudformation` 存储库的 `xray-scorekeep` 文件夹。

```
cd xray-scorekeep/cloudformation/
```

2. 输入以下 AWS CLI 命令来创建 CloudFormation 堆栈：

```
aws cloudformation create-stack --stack-name scorekeep --capabilities
 "CAPABILITY_NAMED_IAM" --template-body file://cf-resources.yaml
```

3. 等到 CloudFormation 堆栈状态变为`CREATE_COMPLETE`，大约需要五分钟。使用以下 AWS CLI 命令检查状态：

```
aws cloudformation describe-stacks --stack-name scorekeep --query
 "Stacks[0].StackStatus"
```

生成跟踪数据

示例应用程序包括一个前端 Web 应用程序。使用 Web 应用程序来生成 API 流量并将跟踪数据发送到 X-Ray。首先，使用 AWS Management Console 或 AWS CLI检索 Web 应用程序 URL：

AWS Management Console

使用查找应用程序 URL AWS Management Console

1. 打开 CloudFormation 控制台
2. 从列表中选择 `scorekeep` 堆栈。
3. 在 `scorekeep` 堆栈页面上选择输出选项卡，然后选择 `LoadBalancerUrl` URL 链接打开 Web 应用程序。

AWS CLI

使用查找应用程序 URL AWS CLI

1. 使用以下命令显示 Web 应用程序的 URL：

```
aws cloudformation describe-stacks --stack-name scorekeep --query
 "Stacks[0].Outputs[0].OutputValue"
```

2. 复制此 URL 并在浏览器中打开以显示 Scorekeep Web 应用程序。

使用 Web 应用程序生成跟踪数据

1. 选择 Create 来创建用户和会话。
2. 键入 game name，将 Rules 设置为 Tic Tac Toe，然后选择 Create 创建一个游戏。
3. 选择 Play 以启动游戏。
4. 选择平铺可进行移动和更改游戏状态。

上述每个步骤都会生成到 API 的 HTTP 请求，并对 DynamoDB 进行下游调用，以读取和写入用户、会话、游戏、移动和状态数据。

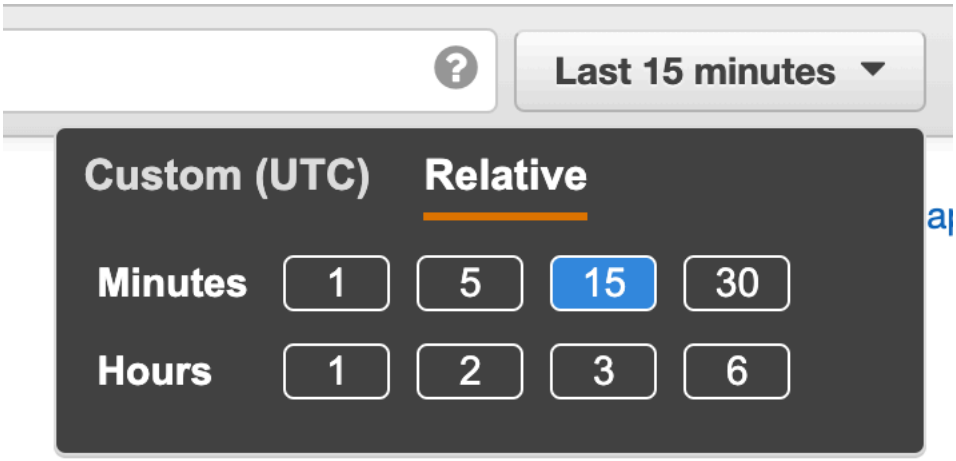
在中查看追踪地图 AWS Management Console

您可以在 X-Ray 和 CloudWatch 控制台中查看示例应用程序生成的轨迹图和轨迹。

X-Ray console

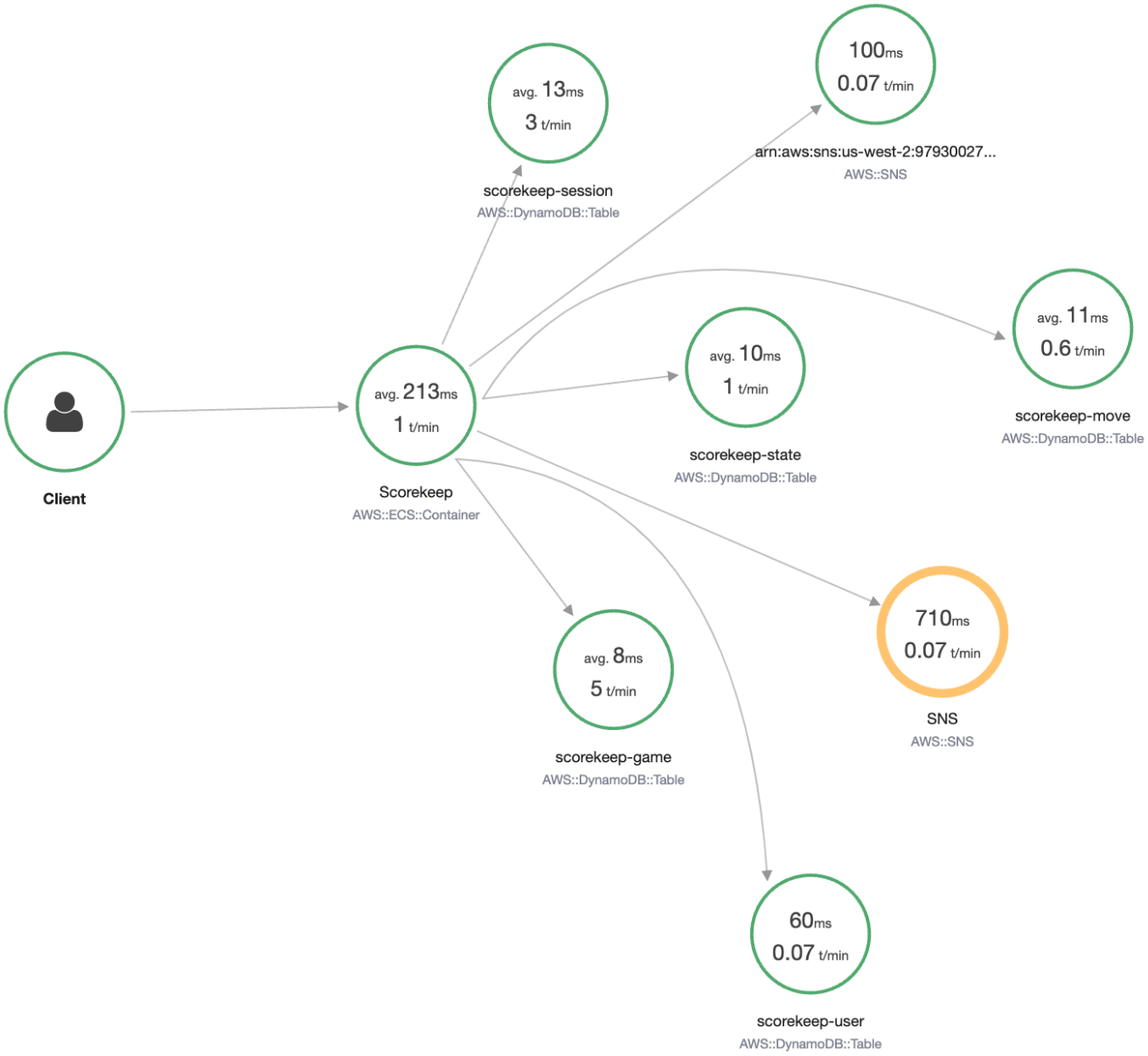
使用 X-Ray 控制台

1. 打开 X-Ray 控制台的跟踪地图页面。
2. 控制台将显示该服务的图形表示形式，这是由 X-Ray 利用应用程序发送的跟踪数据生成的。需要时，务必调整跟踪地图的时间段，以确保将会显示自您首次启动该 Web 应用程序以来的所有跟踪。



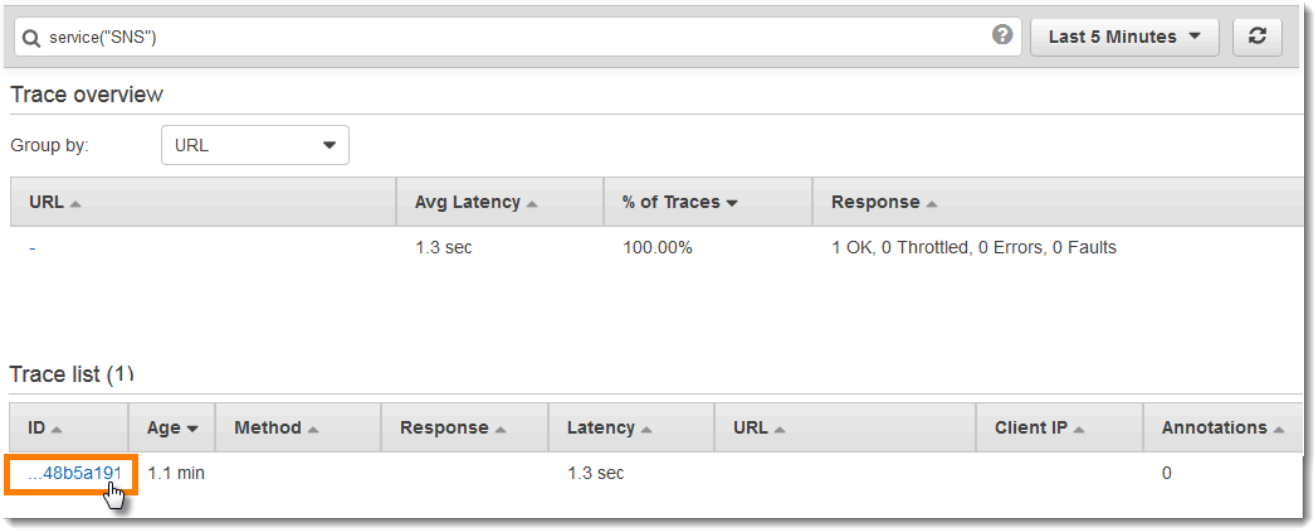
该跟踪地图显示 Web 应用程序客户端、在 Amazon ECS 中运行的 API，以及应用程序使用的每个 DynamoDB 表。对应用程序的每个请求（最多为可配置的每秒最大请求数）都受到跟踪（因为请求到达 API），生成针对下游服务的请求，然后完成。

可以在服务图中选择任一节点，来查看对该节点生成流量的请求的跟踪。目前，Amazon SNS 节点显示为黄色。深入了解原因。

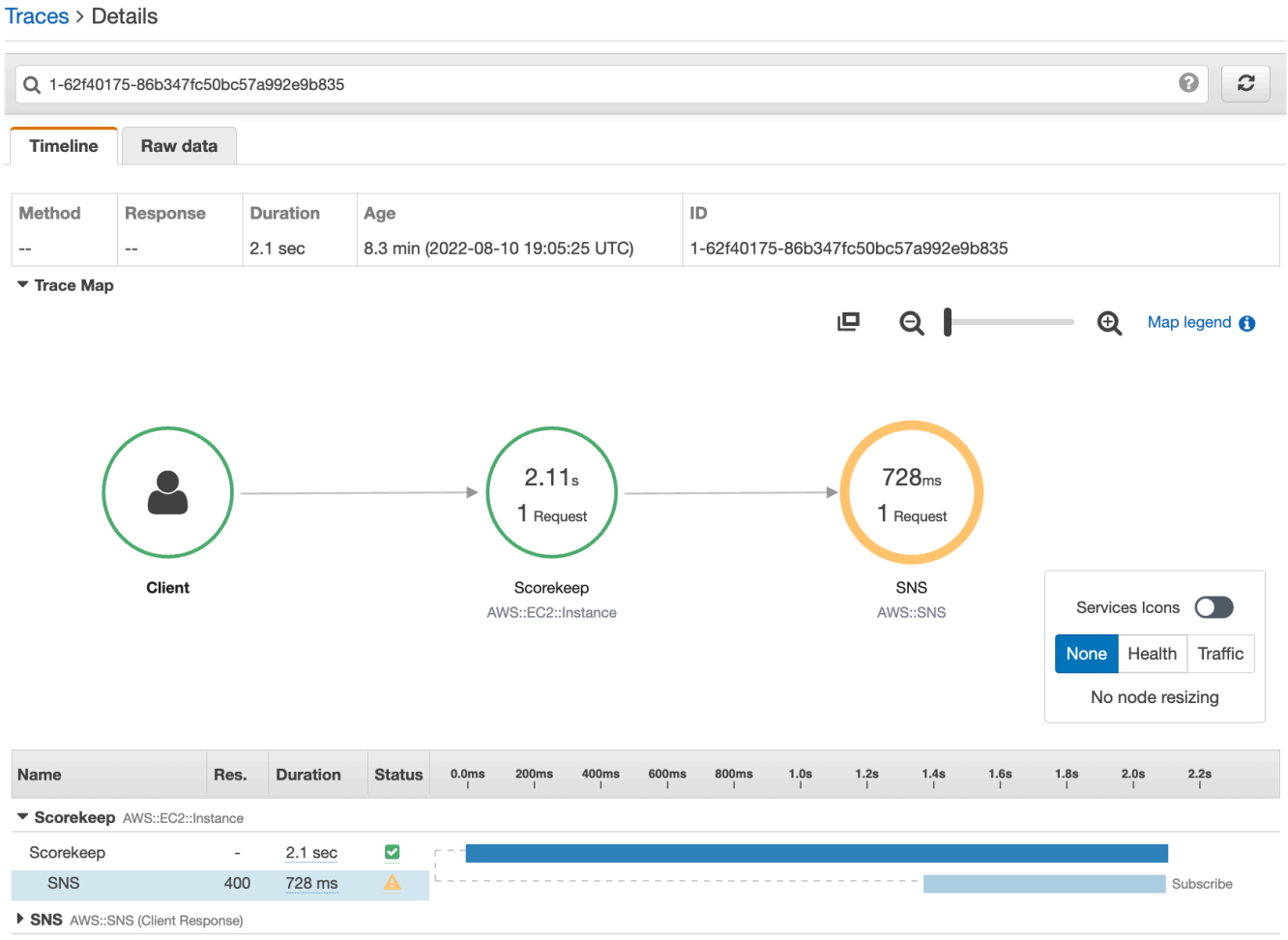


查找错误原因

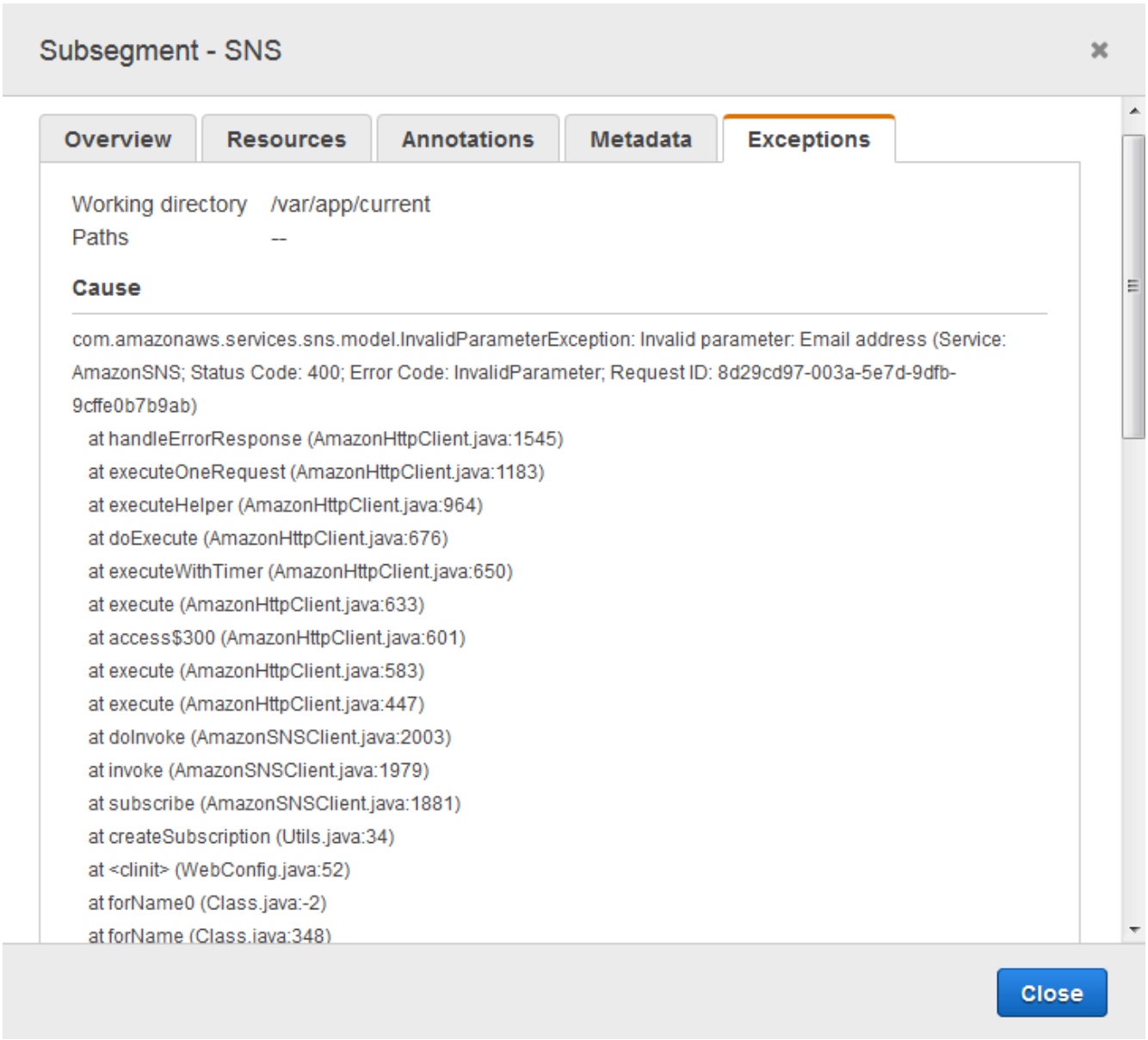
1. 选择名为 SNS 的节点。将会显示该节点的详细信息面板。
2. 选择查看跟踪以访问跟踪概述屏幕。
3. 从跟踪列表中选择跟踪。该跟踪没有方法或 URL，因为它是在启动期间记录的，而不是对传入请求的响应。



4. 选择页面底部 Amazon SNS 分段中的错误状态图标，打开 SNS 子分段的异常页面。



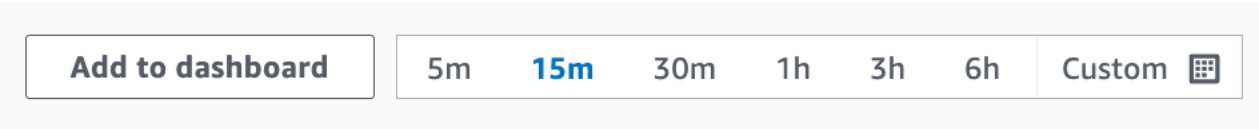
5. X-Ray SDK 会自动捕获由已检测的 AWS SDK 客户端引发的异常并记录堆栈跟踪。



CloudWatch console

使用控制 CloudWatch 台

1. 打开 CloudWatch 控制台的 X-Ray 跟踪地图页面。
2. 控制台将显示该服务的图形表示形式，这是由 X-Ray 利用应用程序发送的跟踪数据生成的。需要时，务必调整跟踪地图的时间段，以确保将会显示自您首次启动该 Web 应用程序以来的所有跟踪。



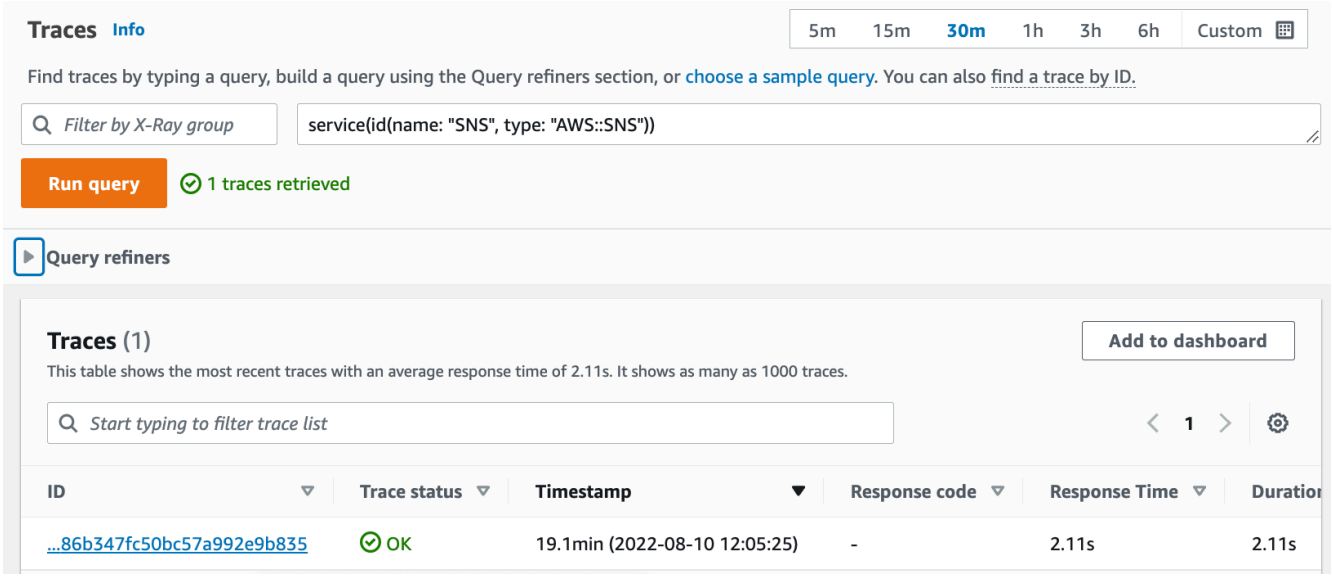
跟踪地图显示了 Web 应用程序客户端、在亚马逊 EC2中运行的 API 以及该应用程序使用的每个 DynamoDB 表。对应用程序的每个请求（最多为可配置的每秒最大请求数）都受到跟踪（因为请求到达 API），生成针对下游服务的请求，然后完成。

可以在服务图中选择任一节点，来查看对该节点生成流量的请求的跟踪。目前，Amazon SNS 节点显示为橙色。深入了解原因。

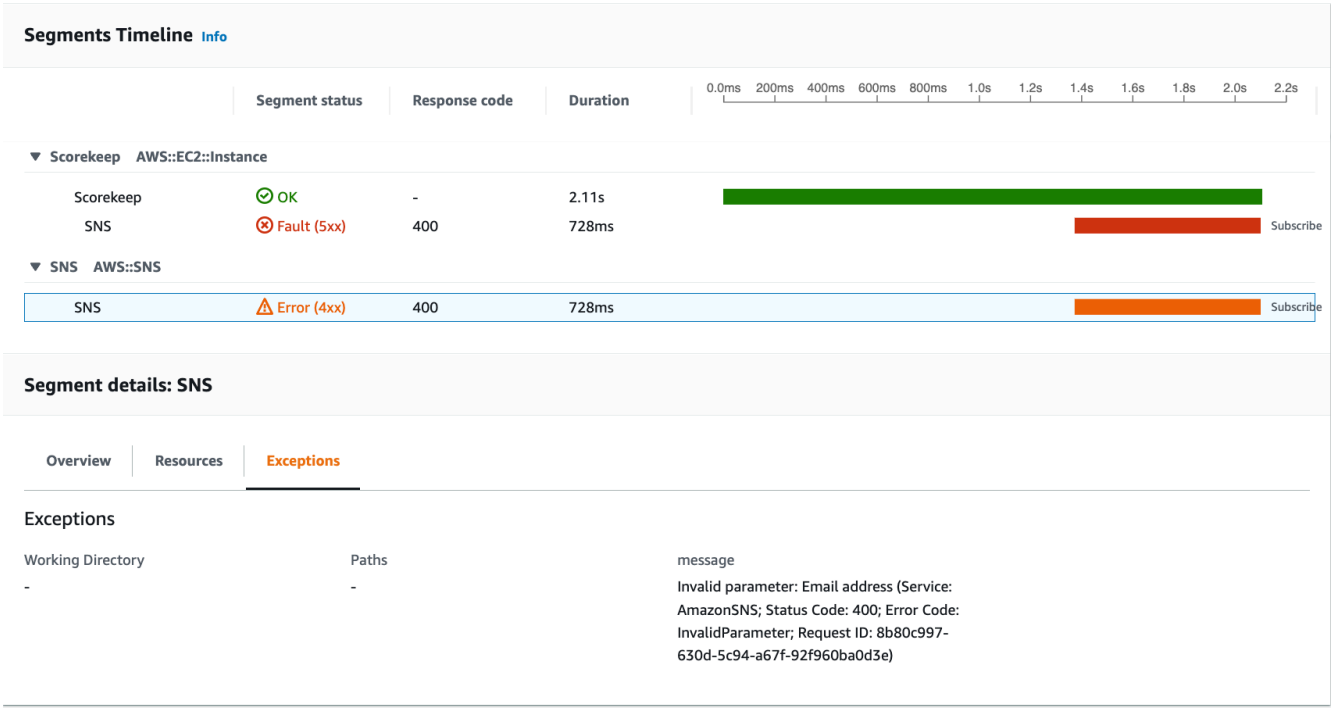

scorekeep-state
DynamoDB Table
scorekeep-game
DynamoDB Table
scorekeep-move
DynamoDB Table
Client
Scorekeep
ECS Container
SNS
SNS
scorekeep-session
DynamoDB Table

查找错误原因

1. 选择名为 SNS 的节点。地图下方显示 SNS 节点详细信息面板。
2. 选择查看跟踪以访问跟踪页面。
3. 添加页面底部，从跟踪列表中选择跟踪。该跟踪没有方法或 URL，因为它是在启动期间记录的，而不是对传入请求的响应。



4. 在分段时间线底部选择 Amazon SNS 子分段，然后选择该 SNS 子分段的异常选项卡以查看异常详细信息。



原因指出，在 `WebConfig` 类中，调用 `createSubscription` 时所提供的电子邮件地址无效。在下一节中，我们将会修复此问题。

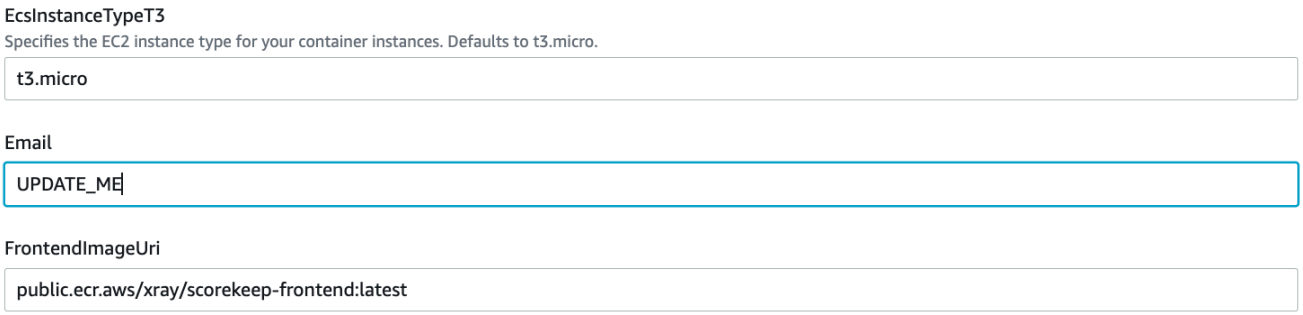
配置 Amazon SNS 通知

当用户完成游戏时，Scorekeep 使用 Amazon SNS 发送通知。当应用程序启动时，它会尝试为 CloudFormation 堆栈参数中定义的电子邮件地址创建订阅。该调用目前会失败。配置通知电子邮件以启用通知，并处理跟踪地图中突出显示的失败。

AWS Management Console

使用配置亚马逊 SNS 通知 AWS Management Console

1. 打开 CloudFormation 控制台
2. 在列表中选择 `scorekeep` 堆栈名称旁边的单选按钮，然后选择更新。
3. 确保选择的是用户当前模板，然后单击更新堆栈页面上的下一步。
4. 在列表中找到电子邮件参数，将默认值替换为有效的电子邮件地址。



5. 滚动到页面底部并选择下一步。
6. 滚动到“查看”页面的底部，选中确认 CloudFormation 可以创建具有自定义名称的 IAM 资源的复选框，然后选择更新堆栈。
7. CloudFormation 堆栈正在更新中。堆栈状态将在五分钟左右保持为 `UPDATE_IN_PROGRESS`，然后变为 `UPDATE_COMPLETE`。状态将会定期更新，也可以刷新页面。

AWS CLI

使用配置亚马逊 SNS 通知 AWS CLI

1. 导航到之前创建的 `xray-scorekeep/cloudformation/` 文件夹，然后在文本编辑器打开 `cf-resources.yaml` 文件。

2. 在“电子邮件”参数中找到该`Default`值，然后将其从更改*`UPDATE_ME`*为有效的电子邮件地址。

```
Parameters:
  Email:
    Type: String
    Default: UPDATE_ME # <- change to a valid abc@def.xyz email address
```

3. 在`cloudformation`文件夹中，使用以下 AWS CLI 命令更新 CloudFormation 堆栈：

```
aws cloudformation update-stack --stack-name scorekeep --capabilities
 "CAPABILITY_NAMED_IAM" --template-body file://cf-resources.yaml
```

4. 等到 CloudFormation 堆栈状态变为`UPDATE_COMPLETE`，这将需要几分钟。使用以下 AWS CLI 命令检查状态：

```
aws cloudformation describe-stacks --stack-name scorekeep --query
 "Stacks[0].StackStatus"
```

更新完成后，Scorekeep 重新启动并创建对 SNS 主题的订阅。当您完成游戏时，检查电子邮件并确认订阅以查看更新。打开跟踪地图，验证对 SNS 的调用不再失败。

浏览应用程序示例

在 Java 中，应用程序示例是一个 HTTP Web API，可配置为使用 X-Ray SDK for Java。当您使用 CloudFormation 模板部署应用程序时，它会创建 DynamoDB 表、Amazon ECS 集群以及在 ECS 上运行 Scorekeep 所需的其他服务。ECS 的任务定义文件是通过创建的 CloudFormation。此文件定义 ECS 集群中每项任务使用的容器映像。这些映像从官方 X-Ray 公共 ECR 中获取。Scorekeep API 容器映像具有兼容 Gradle 的 API。Scorekeeep 前端容器的容器映像充当使用 nginx 代理服务器的前端。此服务器会将传送到以 /api 开头的路径的请求路由到 API。

要检测传入 HTTP 请求，应用程序将添加 SDK 提供的 `TracingFilter`。

Example src/main/java/scorekeep/WebConfig.java-servlet 过滤器

```
import javax.servlet.Filter;
import com.amazonaws.xray.javax.servlet.AWSXRayServletFilter;
...

@Configuration
```

```
public class WebConfig {

  @Bean
  public Filter TracingFilter() {
    return new AWSXRayServletFilter("Scorekeep");
  }
...
```

此筛选器会发送有关应用程序所处理所有传入请求的跟踪数据，包括请求 URL、方法、响应状态、开始时间和结束时间。

应用程序还会使用 适用于 Java 的 AWS SDK对 DynamoDB 进行下游调用。要检测这些调用，应用程序只需将与 AWS SDK 相关的子模块作为依赖项，适用于 Java 的 X-Ray SDK 会自动检测所有 AWS SDK 客户端。

应用程序使用 `Docker` 在实例上生成源代码，使用 `Gradle Docker Image` 和 `Scorekeep API Dockerfile` 文件运行 Gradle 在其 `ENTRYPOINT` 生成的可执行 JAR。

Example 使用 Docker 通过 Gradle Docker 映像进行构建

```
docker run --rm -v /PATH/TO/SCOREKEEP_REPO/home/gradle/project -w /home/gradle/project
 gradle:4.3 gradle build
```

Example Dockerfile ENTRYPOINT

```
ENTRYPOINT [ "sh", "-c", "java -Dserver.port=5000 -jar scorekeep-api-1.0.0.jar" ]
```

在编译期间，`build.gradle` 从 Maven 下载 SDK 子模块，方法是将这些子模块声明为依赖项。

Example build.gradle - 依赖项

```
...
dependencies {
    compile("org.springframework.boot:spring-boot-starter-web")
    testCompile('org.springframework.boot:spring-boot-starter-test')
    compile('com.amazonaws:aws-java-sdk-dynamodb')
    compile("com.amazonaws:aws-xray-recorder-sdk-core")
    compile("com.amazonaws:aws-xray-recorder-sdk-aws-sdk")
    compile("com.amazonaws:aws-xray-recorder-sdk-aws-sdk-instrumentor")
    ...
}
dependencyManagement {
```

```
    imports {
        mavenBom("com.amazonaws:aws-java-sdk-bom:1.11.67")
        mavenBom("com.amazonaws:aws-xray-recorder-sdk-bom:2.11.0")
    }
}
```

核心、 AWS SDK 和 S AWS DK Instrumentor 子模块就是自动检测使用 SDK 进行的任何下游调用所 AWS 必需的。

如需将原始分段数据中断到 X-Ray API，则需要使用 X-Ray 进程守护程序侦听流量或 UDP 端口 2000。为此，应用程序让 X-Ray 进程守护程序在 ECS 上作为附加容器与 Scorekeep 应用程序一起部署的容器中运行。请参阅 X-Ray 进程守护程序主题了解更多信息。

Example ECS 任务定义中的 X-Ray 进程守护程序容器定义。

```
...
Resources:
  ScorekeepTaskDefinition:
    Type: AWS::ECS::TaskDefinition
    Properties:
      ContainerDefinitions:
      ...

      - Cpu: '256'
        Essential: true
        Image: amazon/aws-xray-daemon
        MemoryReservation: '128'
        Name: xray-daemon
        PortMappings:
          - ContainerPort: '2000'
            HostPort: '2000'
            Protocol: udp
      ...
```

X-Ray SDK for Java 提供了一个名为 `AWSXRay` 的类，该类提供全局记录器，即您可用于检测代码的 `TracingHandler`。您可以配置全局记录器以自定义为传入 HTTP 调用创建分段的 `AWSXRayServletFilter`。示例包括 `WebConfig` 类中的一个静态数据块，该数据块使用插件和示例规则配置全局记录器。

Example src/main/java/scorekeep/WebConfig.java-录音机

```
import com.amazonaws.xray.AWSXRay;
```

```
import com.amazonaws.xray.AWSXRayRecorderBuilder;
import com.amazonaws.xray.javax.servlet.AWSXRayServletFilter;
import com.amazonaws.xray.plugins.ECSPlugin;
import com.amazonaws.xray.plugins.EC2Plugin;
import com.amazonaws.xray.strategy.sampling.LocalizedSamplingStrategy;
...

@Configuration
public class WebConfig {
  ...

  static {
    AWSXRayRecorderBuilder builder = AWSXRayRecorderBuilder.standard().withPlugin(new
 ECSPlugin()).withPlugin(new EC2Plugin());

    URL ruleFile = WebConfig.class.getResource("/sampling-rules.json");
    builder.withSamplingStrategy(new LocalizedSamplingStrategy(ruleFile));

    AWSXRay.setGlobalRecorder(builder.build());
    ...

  }
}
```

该示例使用生成器加载来自名为 `sampling-rules.json` 的文件的采样规则。采样规则确定 SDK 记录传入请求分段的速率。

Example src/main/java/resources/sampling-rules.json

```
{
  "version": 1,
  "rules": [
    {
      "description": "Resource creation.",
      "service_name": "*",
      "http_method": "POST",
      "url_path": "/api/*",
      "fixed_target": 1,
      "rate": 1.0
    },
    {
      "description": "Session polling.",
      "service_name": "*",
```

```
      "http_method": "GET",
      "url_path": "/api/session/*",
      "fixed_target": 0,
      "rate": 0.05
    },
    {
      "description": "Game polling.",
      "service_name": "*",
      "http_method": "GET",
      "url_path": "/api/game/*/*",
      "fixed_target": 0,
      "rate": 0.05
    },
    {
      "description": "State polling.",
      "service_name": "*",
      "http_method": "GET",
      "url_path": "/api/state/*/*/*",
      "fixed_target": 0,
      "rate": 0.05
    }
  ],
  "default": {
    "fixed_target": 1,
    "rate": 0.1
  }
}
```

采样规则文件定义了四个自定义采样规则和默认规则。对于每个传入请求，SDK 按定义的顺序评估自定义规则。SDK 应用与请求的方法、路径和服务名称匹配的第一个规则。对于 Scorekeep，第一个规则通过应用每秒 1 个请求的固定目标和 1.0 的速率来捕获所有 POST 请求 (资源创建调用)，或者，在满足固定目标后，捕获 100% 的请求。

另外三个自定义规则应用 5% 的速率，对于会话、游戏和状态读取无固定目标（GET 请求）。这样可以尽可能减少前端为确保内容最新而每隔几秒钟自动发出的定期调用的跟踪数。对于所有其他请求，该文件定义默认速率为每秒 1 个请求，速率为 10%。

示例应用程序还展示了如何使用高级特征，如手动 SDK 客户端检测、创建其他子分段和传出 HTTP 调用。有关更多信息，请参阅 AWS X-Ray 示例应用程序。

可选：最低权限策略

Scorekeeep ECS 容器使用 `AmazonSNSFullAccess` 和 `AmazonDynamoDBFullAccess` 等完整访问策略来访问资源。对于生产应用程序而言，使用完整访问策略并不是最佳做法。以下示例更新 DynamoDB IAM 策略以提升应用程序的安全性。要详细了解 IAM 策略中的安全最佳实践，请参阅 AWS X-Ray 的身份和访问管理。

Example cf-resources.yaml 模板角色定义 ECSTask

```
ECSTaskRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
          -
            Effect: "Allow"
            Principal:
              Service:
                - "ecs-tasks.amazonaws.com"
            Action:
              - "sts:AssumeRole"
      ManagedPolicyArns:
        - "arn:aws:iam::aws:policy/AmazonDynamoDBFullAccess"
        - "arn:aws:iam::aws:policy/AmazonSNSFullAccess"
        - "arn:aws:iam::aws:policy/AWSXrayFullAccess"
      RoleName: "scorekeepRole"
```

要更新策略，您首先需要确定 DynamoDB 资源的 ARN。然后，使用自定义 IAM 策略中的 ARN。最后，将该策略应用到实例配置文件。

如何识别 DynamoDB 资源的 ARN：

1. 打开 DynamoDB 控制台。
2. 从左侧导航栏中选择表。
3. 选择任意一个 `scorekeep-*` 显示表的详细信息页面。
4. 在概述选项卡下，选择其他信息展开此部分，查看 Amazon 资源名称（ARN）。复制该值。
5. 将 ARN 插入到以下 IAM 策略中，将 `AWS_REGION` 替换为具有您的具体区域和账户 ID 的 `AWS_ACCOUNT_ID` 值。此新策略仅允许执行指定的操作，而非允许执行任何操作的 `AmazonDynamoDBFullAccess` 策略。

Example

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ScorekeepDynamoDB",
            "Effect": "Allow",
            "Action": [
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem",
                "dynamodb:Scan",
                "dynamodb:Query"
            ],
            "Resource": "arn:aws:dynamodb:<AWS_REGION>:<AWS_ACCOUNT_ID>:table/
scorekeep-*"
        }
    ]
}
```

应用程序创建的表遵循一致的命名约定。可以使用 `scorekeep-*` 格式指示所有 Scorekeep 表。

更改 IAM 策略

1. 从 IAM 控制台打开 Scorekeep 任务角色 (scorekeepRole)。
2. 选择 `AmazonDynamoDBFullAccess` 策略旁边的复选框，然后选择删除以删除此策略。
3. 选择添加权限，然后选择附加策略，最后选择创建策略。
4. 选择 JSON 选项卡，然后粘贴上面创建的策略。
5. 在页面底部，选择下一步：标签。
6. 在页面底部，选择下一步：查看。
7. 在名称中，为策略分配一个名称。
8. 在页面底部，选择创建策略。
9. 将新创建的策略附加到 `scorekeepRole` 角色。附加的策略更改可能需要几分钟才能生效。

如果您已将新策略附加到该scorekeepRole角色，则必须在删除 CloudFormation 堆栈之前将其分离，因为此附加的策略将阻止堆栈被删除。删除此策略即可自动附加此策略。

删除自定义 IAM 策略

1. 打开 IAM 管理控制台。
2. 从左侧导航菜单中，选择策略。
3. 搜索在本节早些时候创建的自定义策略，然后选择策略名称旁边的单选按钮以突出显示它。
4. 选择操作下拉列表，然后选择删除。
5. 键入自定义策略的名称，然后选择删除以确认删除。此操作将会自动取消附加 scorekeepRole 角色中的策略。

清理

请按照以下步骤删除 Scorekeep 应用程序资源：

> **Note**
>
> 如果您使用本教程的前一部分创建并附加了自定义策略，则必须先从中移除策略，scorekeepRole然后才能删除 CloudFormation 堆栈。

AWS Management Console

使用删除示例应用程序 AWS Management Console

1. 打开 CloudFormation 控制台
2. 在列表中选择 scorekeep 堆栈名称旁边的单选按钮，然后选择删除。
3. CloudFormation 堆栈现在正在被删除。堆栈状态将会在几分钟内保持为 DELETE_IN_PROGRESS，直到所有资源被删除。状态将会定期更新，也可以刷新页面。

AWS CLI

使用删除示例应用程序 AWS CLI

1. 输入以下 AWS CLI 命令删除 CloudFormation 堆栈：

```
aws cloudformation delete-stack --stack-name scorekeep
```

2. 等到 CloudFormation 堆栈不复存在，这大约需要五分钟。使用以下 AWS CLI 命令检查状态：

```
aws cloudformation describe-stacks --stack-name scorekeep --query
 "Stacks[0].StackStatus"
```

后续步骤

要了解有关 X-Ray 的更多信息，请参阅下一章AWS X-Ray 概念。

要测试你自己的应用程序，请详细了解适用于 Java 的 X-Ray SDK 或其他 X-Ray 中的一个 SDKs：

- X-Ray SDK for Java - AWS X-Ray 适用于 Java 的 SDK
- X-Ray SDK for Node.js - AWS 适用于 Node.js 的 X-ray SDK
- X-Ray SDK for .NET - AWS X-Ray 适用于.NET 的 SDK

要在本地或在本地运行 X-Ray 守护程序 AWS，请参阅AWS X-Ray 守护程序。

要为上的示例应用程序做出贡献 GitHub，请参阅eb-java-scorekeep。

手动检测 S AWS DK 客户端

当你在构建依赖项中包含 SDK Instrumentor 子模块时， AWS 适用于 Java 的 X-Ray AWS SDK 会自动检测所有 SDK 客户端。

您可以通过删除 Instrumentor 子模块来禁用自动客户端检测。这使您可以手动检测一些客户端而忽略另一些客户端，或者在不同客户端上使用不同跟踪处理程序。

为了说明对检测特定 S AWS DK 客户端的支持，应用程序将跟踪处理程序`AmazonDynamoDBClientBuilder`作为用户、游戏和会话模型中的请求处理程序传递给。此代码更改告知 SDK 使用这些客户端检测对 DynamoDB 的所有调用。

Example **`src/main/java/scorekeep/SessionModel.java`** - 手动 AWS SDK 客户端检测

```
import com.amazonaws.xray.AWSXRay;
import com.amazonaws.xray.handlers.TracingHandler;
```

```
public class SessionModel {
  private AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard()
        .withRegion(Constants.REGION)
        .withRequestHandlers(new TracingHandler(AWSXRay.getGlobalRecorder()))
        .build();
  private DynamoDBMapper mapper = new DynamoDBMapper(client);
```

如果您从项目依赖项中移除 AWS SDK Instrumentor 子模块，则只有手动检测的 AWS SDK 客户端才会出现在跟踪图中。

创建附加子分段

在用户模型类中，应用程序需要手动创建子分段，以便对 `saveUser` 函数中执行的所有下游调用进行分组和添加元数据。

Example **src/main/java/scorekeep/UserModel.java** - 自定义子分段

```
import com.amazonaws.xray.AWSXRay;
import com.amazonaws.xray.entities.Subsegment;
...
    public void saveUser(User user) {
    // Wrap in subsegment
    Subsegment subsegment = AWSXRay.beginSubsegment("## UserModel.saveUser");
    try {
      mapper.save(user);
    } catch (Exception e) {
      subsegment.addException(e);
      throw e;
    } finally {
      AWSXRay.endSubsegment();
    }
  }
```

记录注释、元数据和用户 IDs

在游戏模型类中，每当应用程序将游戏保存在 DynamoDB 中时，都会将 Game 对象记录到元数据块中。另外，该应用程序将游戏记录 IDs 在注释中，以便与过滤器表达式一起使用。

Example **src/main/java/scorekeep/GameModel.java** - 注释和元数据

```
import com.amazonaws.xray.AWSXRay;
```

```
import com.amazonaws.xray.entities.Segment;
import com.amazonaws.xray.entities.Subsegment;
...
  public void saveGame(Game game) throws SessionNotFoundException {
    // wrap in subsegment
    Subsegment subsegment = AWSXRay.beginSubsegment("## GameModel.saveGame");
    try {
      // check session
      String sessionId = game.getSession();
      if (sessionModel.loadSession(sessionId) == null ) {
        throw new SessionNotFoundException(sessionId);
      }
      Segment segment = AWSXRay.getCurrentSegment();
      subsegment.putMetadata("resources", "game", game);
      segment.putAnnotation("gameid", game.getId());
      mapper.save(game);
    } catch (Exception e) {
      subsegment.addException(e);
      throw e;
    } finally {
      AWSXRay.endSubsegment();
    }
  }
```

在移动控制器中，应用程序 IDs使用记录用户setUser。用户 IDs被记录在区段的单独字段中，并编制索引以供搜索使用。

Example src/main/java/scorekeep/MoveController.java — 用户 ID

```
import com.amazonaws.xray.AWSXRay;
...
  @RequestMapping(value="/{userId}", method=RequestMethod.POST)
  public Move newMove(@PathVariable String sessionId, @PathVariable String
 gameId, @PathVariable String userId, @RequestBody String move) throws
 SessionNotFoundException, GameNotFoundException, StateNotFoundException,
 RulesException {
    AWSXRay.getCurrentSegment().setUser(userId);
    return moveFactory.newMove(sessionId, gameId, userId, move);
  }
```

检测传出 HTTP 调用

用户工厂类显示应用程序如何使用 X-Ray SDK for Java 的 HTTPClientBuilder 版本来检测传出 HTTP 调用。

Example **src/main/java/scorekeep/UserFactory.java**— HTTPClient 仪器

```
import com.amazonaws.xray.proxies.apache.http.HttpClientBuilder;

  public String randomName() throws IOException {
    CloseableHttpClient httpclient = HttpClientBuilder.create().build();
    HttpGet httpGet = new HttpGet("http://uinames.com/api/");
    CloseableHttpResponse response = httpclient.execute(httpGet);
    try {
      HttpEntity entity = response.getEntity();
      InputStream inputStream = entity.getContent();
      ObjectMapper mapper = new ObjectMapper();
      Map<String, String> jsonMap = mapper.readValue(inputStream, Map.class);
      String name = jsonMap.get("name");
      EntityUtils.consume(entity);
      return name;
    } finally {
      response.close();
    }
  }
```

如果您当前使用 `org.apache.http.impl.client.HttpClientBuilder`，则只需使用 `com.amazonaws.xray.proxies.apache.http.HttpClientBuilder` 的语句换出该类的导出语句。

检测对 PostgreSQL 数据库的调用

`application-pgsql.properties` 文件将 X-Ray PostgreSQL 跟踪拦截程序添加到在 `RdsWebConfig.java` 中创建的数据源。

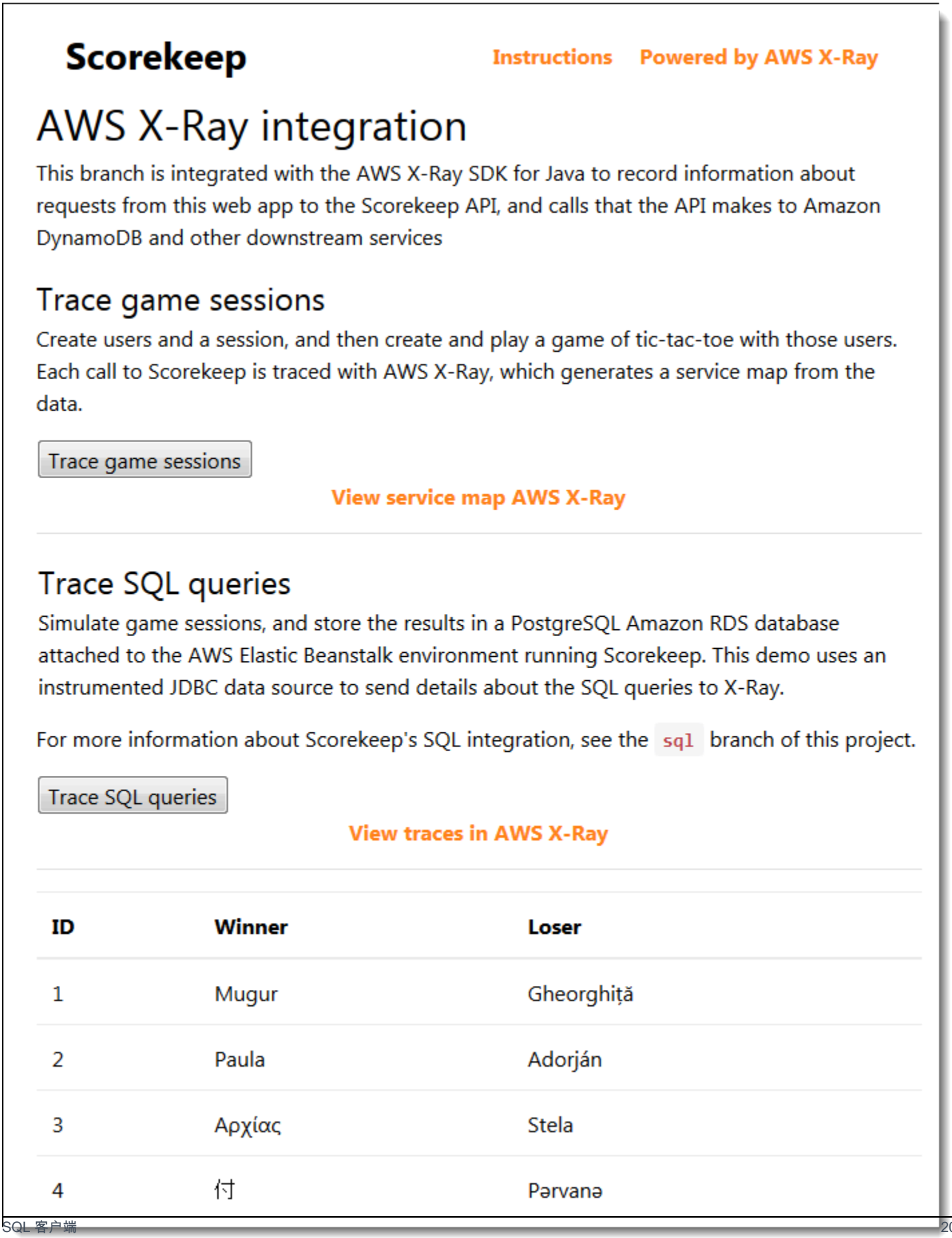
Example **application-pgsql.properties** - PostgreSQL 数据库检测

```
spring.datasource.continue-on-error=true
spring.jpa.show-sql=false
spring.jpa.hibernate.ddl-auto=create-drop
```

```
spring.datasource.jdbc-interceptors=com.amazonaws.xray.sql.postgres.TracingInterceptor
spring.jpa.database-platform=org.hibernate.dialect.PostgreSQL94Dialect
```

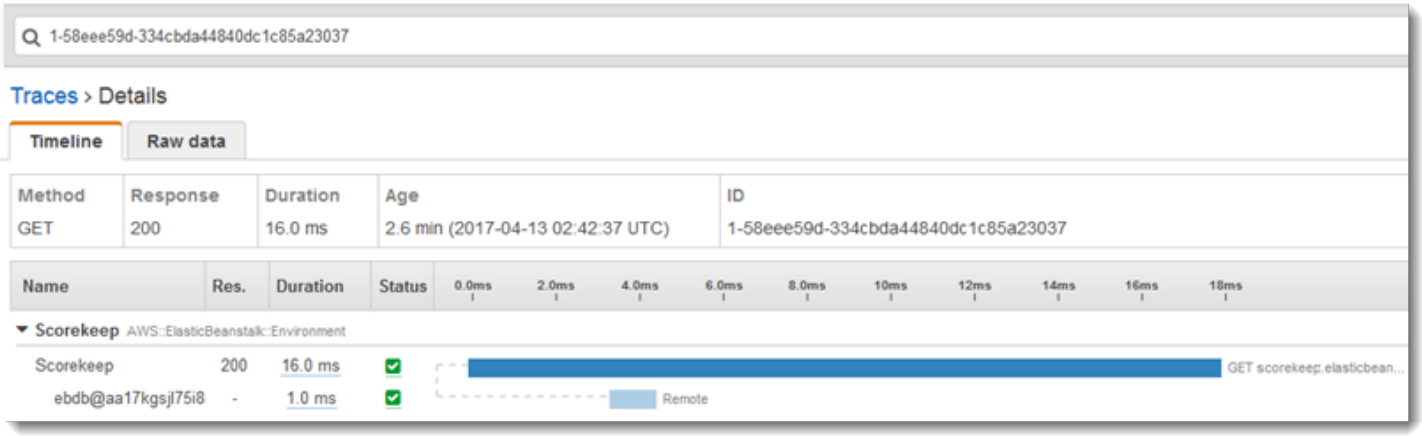
> **Note**
>
> 有关如何将 PostgreSQL 数据库添加到应用程序环境的详细信息，请参阅 https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.managing.db.html 开发人员指南 中的AWS Elastic Beanstalk 使用 Elastic Beanstalk 配置数据库。

`xray` 分支中的 X-Ray 演示页包含一个使用检测的数据源生成跟踪的演示，此跟踪显示有关其生成的 SQL 查询的信息。导航到正在运行的应用程序中的 `/#/xray` 路径，或选择导航栏中的 Powered by AWS X-Ray 查看该演示页。


Scorekeep
Instructions
Powered by AWS X-Ray
AWS X-Ray integration
This branch is integrated with the AWS X-Ray SDK for Java to record information about requests from this web app to the Scorekeep API, and calls that the API makes to Amazon DynamoDB and other downstream services
Trace game sessions
Create users and a session, and then create and play a game of tic-tac-toe with those users. Each call to Scorekeep is traced with AWS X-Ray, which generates a service map from the data.
Trace game sessions
View service map AWS X-Ray
Trace SQL queries
Simulate game sessions, and store the results in a PostgreSQL Amazon RDS database attached to the AWS Elastic Beanstalk environment running Scorekeep. This demo uses an instrumented JDBC data source to send details about the SQL queries to X-Ray.
For more information about Scorekeep's SQL integration, see the sql branch of this project.
Trace SQL queries
View traces in AWS X-Ray
ID
Winner
Loser
1
Mugur
Gheorghiță
2
Paula
Adorján
3
Αρχίας
Stela
4
付
Pərvanə

选择 Trace SQL queries 模拟游戏会话并将结果存储在附加的数据库中。然后，选择 “在 AWS X-Ray 中查看跟踪”，查看经过筛选的到达该 API `/api/history` 路线的跟踪列表。

从该列表中选择一个跟踪以查看时间线，包括 SQL 查询。



仪表函数 AWS Lambda

Scorekeep 使用两个 AWS Lambda 函数。第一个是来自 `lambda` 分支的 Node.js 函数，它为新用户生成随机名称。如果用户在创建会话时未输入名称，则该应用程序将通过 适用于 Java 的 AWS SDK调用名为 `random-name` 的函数。适用于 Java 的 X-Ray SDK 在子分段中记录有关对 Lambda 的调用的信息，就像使用仪器 AWS 化的 SDK 客户端进行的任何其他调用一样。

> **Note**
>
> 运行 `random-name` Lambda 函数需要在 Elastic Beanstalk 环境外创建其他资源。有关详细信息和说明，请参阅自述文件：AWS Lambda 集成。

第二个函数为 `scorekeep-worker`，它是一个独立于 Scorekeep API 运行的 Python 函数。当游戏结束时，API 将会话 ID 和游戏 ID 写入 SQS 队列。工作线程函数将从队列中读取项目，然后调用 Scorekeep API 来为 Amazon S3 中的存储构建每个游戏会话的完整记录。

Scorekeep 包括用于创建这两个函数的 AWS CloudFormation 模板和脚本。由于您需要将 X-Ray SDK 与函数代码绑定，因此，这些模板无需任何代码即可创建函数。在部署 Scorekeep 时，`.ebextensions` 文件夹中包含的配置文件将创建一个包含 SDK 的源包并使用 AWS Command Line Interface更新函数代码和配置。

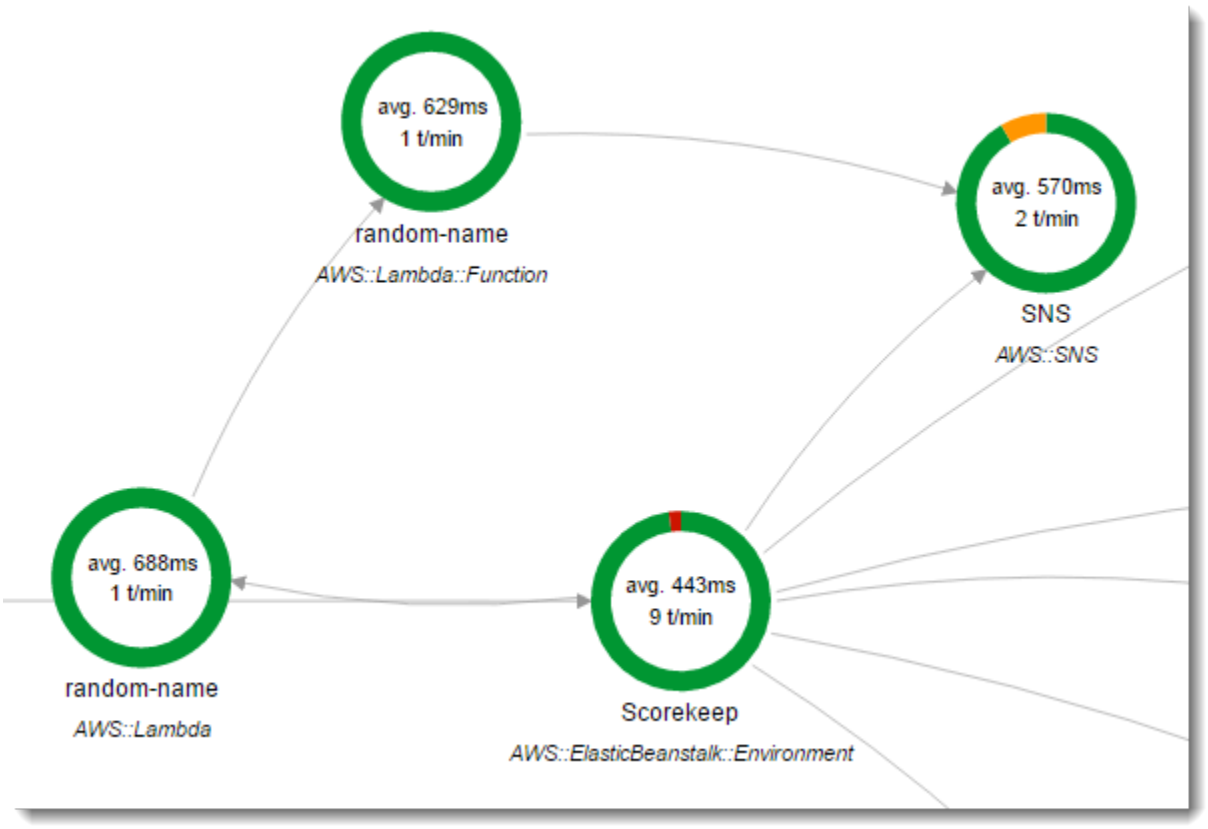
函数

随机名称

当用户在没有登录或指定用户名的情况下启动游戏会话时，Scorekeep 将调用随机名称函数。当 Lambda 处理对 `random-name` 的调用时，它读取跟踪标头，其中包含 X-Ray SDK for Java 写入的跟踪 ID 和采样决策。

对于每个被采样的请求，Lambda 运行 X-Ray 进程守护程序并编写两个分段。第一个分段记录有关调用函数的 Lambda 调用的信息。但从 Lambda 的角度看，该分段包含与 Scorekeep 记录的子分段相同的信息。第二个分段表示函数所做的工作。

Lambda 通过函数上下文将函数分段传递到 X-Ray SDK。在检测 Lambda 函数时，您不使用 SDK 为传入请求创建分段。Lambda 将提供分段，并且您将使用 SDK 检测客户端和写入子分段。



`random-name` 函数在 Node.js 中实现。它使用 Node.js JavaScript 中的 SDK 通过亚马逊 SNS 发送通知，使用 Node.js 的 X-Ray SDK 来检测 S AWS DK 客户端。为了写入注释，该函数利用

AWSXRay.captureFunc 创建一个自定义子分段，并在经过检测的函数中写入注释。在 Lambda 中，您无法直接将注释写入函数分段，而只能将其写入您创建的子分段。

Example **function/index.js** -- 随机名称 Lambda 函数

```
var AWSXRay = require('aws-xray-sdk-core');
var AWS = AWSXRay.captureAWS(require('aws-sdk'));

AWS.config.update({region: process.env.AWS_REGION});
var Chance = require('chance');

var myFunction = function(event, context, callback) {
  var sns = new AWS.SNS();
  var chance = new Chance();
  var userid = event.userid;
  var name = chance.first();

  AWSXRay.captureFunc('annotations', function(subsegment){
    subsegment.addAnnotation('Name', name);
    subsegment.addAnnotation('UserID', event.userid);
  });

  // Notify
  var params = {
    Message: 'Created randon name "' + name + '"" for user "' + userid + '".',
    Subject: 'New user: ' + name,
    TopicArn: process.env.TOPIC_ARN
  };
  sns.publish(params, function(err, data) {
    if (err) {
      console.log(err, err.stack);
      callback(err);
    }
    else {
      console.log(data);
      callback(null, {"name": name});
    }
  });
};

exports.handler = myFunction;
```

在您将示例应用程序部署到 Elastic Beanstalk 时，将自动创建此函数。`xray` 分支包括一个用于创建空白 Lambda 函数的脚本。`.ebextensions`文件夹中的配置文件在部署`npm install`期间使用构建函数包，然后使用 CLI AWS 更新 Lambda 函数。

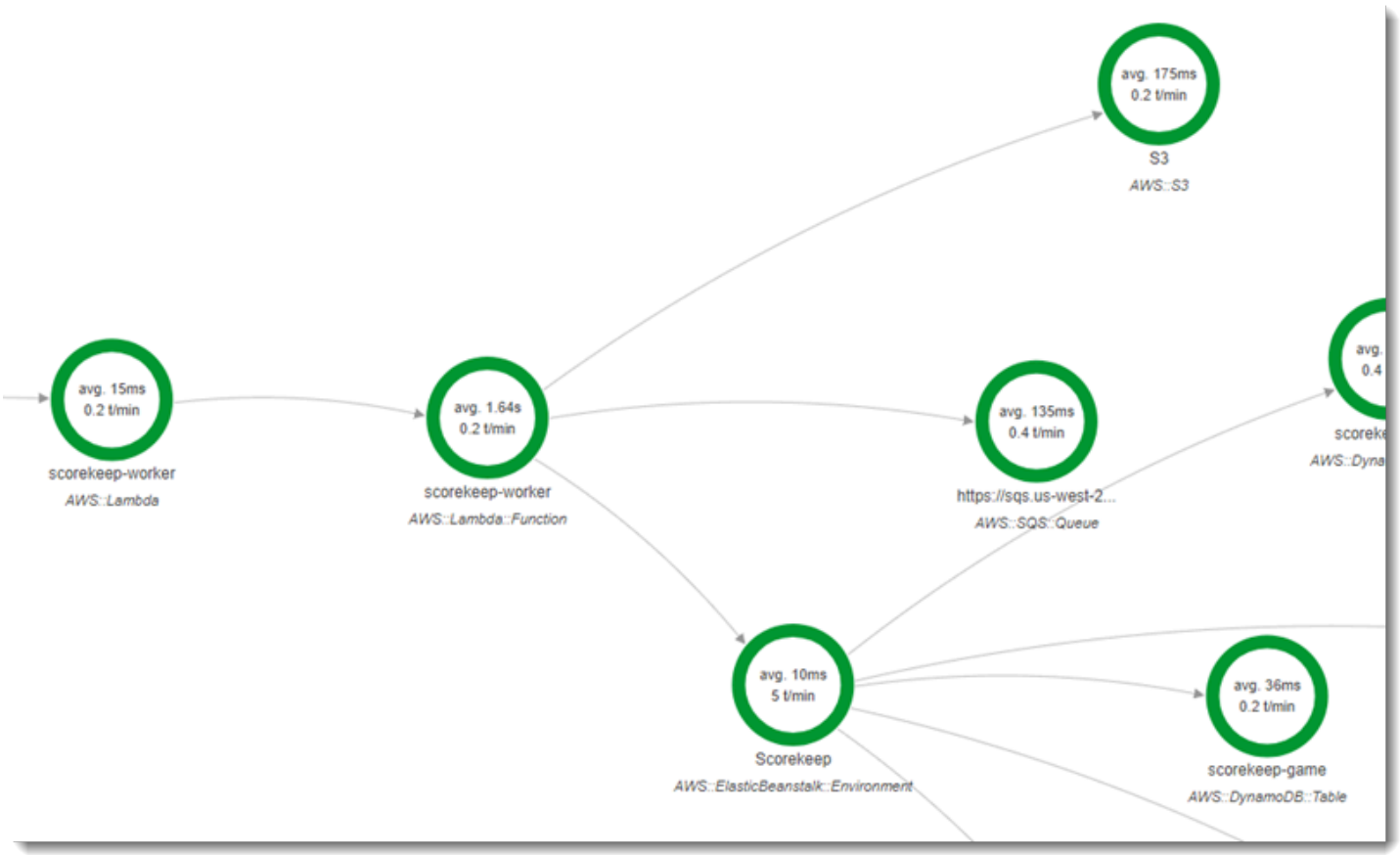
工作线程

经过检测的工作线程函数在自己的分支 `xray-worker` 中提供，这是因为，除非您先创建工作线程函数和相关资源，否则该函数无法运行。有关说明，请参阅分支自述文件。

该函数由每 5 分钟一次捆绑的 Amazon Events CloudWatch 事件触发。当该函数运行时，它会从 Scorekeep 管理的 Amazon SQS 队列中拉取项目。每条消息均包含有关已完成游戏的信息。

工作线程将从游戏记录引用的其他表中拉取游戏记录和文档。例如，DynamoDB 中的游戏记录包含在游戏期间执行的移动的列表。该列表不包含动作本身，而是 IDs 存储在单独表中的移动。

会话和状态也将存储为引用。虽然这可阻止游戏表中的条目过大，但需要额外调用来获取有关游戏的所有信息。工作线程会取消引用所有这些条目，并将游戏的完整记录构建为 Amazon S3 中的单一文档。当您要对数据进行分析时，您可以利用 Amazon Athena 直接在 Amazon S3 中对数据运行查询，而无需运行读取密集型数据迁移来拉取 DynamoDB 中的数据。



工作线程函数已在自身在 AWS Lambda的配置中启用活动跟踪。与随机命名函数不同，worker 不会收到来自已检测应用程序的请求，因此 AWS Lambda 不会收到跟踪标头。利用活动跟踪，Lambda 将创建跟踪 ID 并制定采样决策。

适用于 Python 的 X-Ray SDK 只是函数顶部的几行，用于导入软件开发工具包并运行其`patch_all`函数来修补 HTTclients ，它用来调用 Amazon SQS 和 Amazon S3。 AWS SDK for Python (Boto) 当工作线程调用 Scorekeep API 时，SDK 会将跟踪标头添加到通过 API 跟踪调用的请求中。

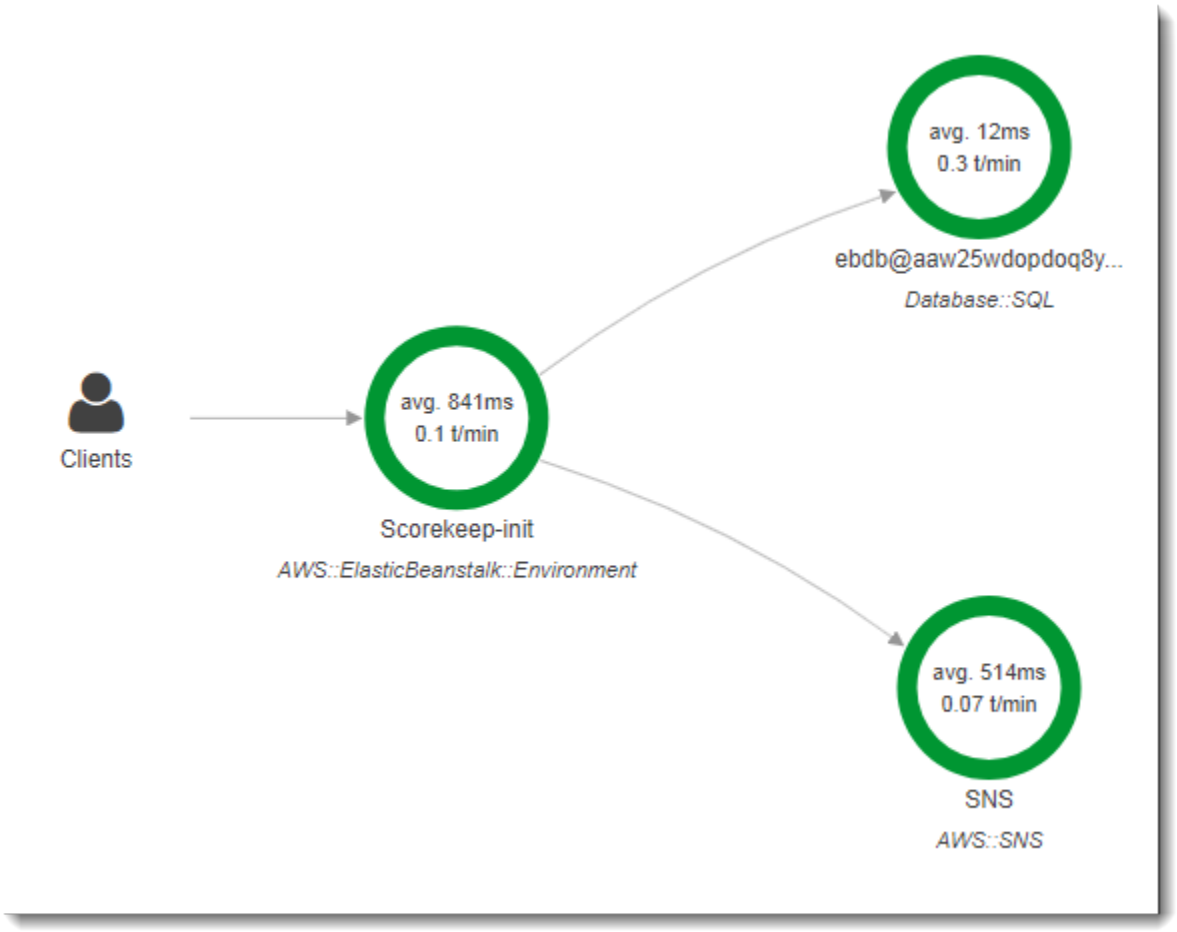
Example **_lambda/scorekeep-worker/scorekeep-worker.py** -- 工作线程 Lambda 函数

```
import os
import boto3
import json
import requests
import time
from aws_xray_sdk.core import xray_recorder
from aws_xray_sdk.core import patch_all

patch_all()
queue_url = os.environ['WORKER_QUEUE']

def lambda_handler(event, context):
    # Create SQS client
    sqs = boto3.client('sqs')
    s3client = boto3.client('s3')

    # Receive message from SQS queue
    response = sqs.receive_message(
        QueueUrl=queue_url,
        AttributeNames=[
            'SentTimestamp'
        ],
        MaxNumberOfMessages=1,
        MessageAttributeNames=[
            'All'
        ],
        VisibilityTimeout=0,
        WaitTimeSeconds=0
    )
   ...
```

检测启动代码

适用于 Java 的 X-Ray 开发工具包自动为传入请求创建分段。只要请求在范围内，您就可以使用检测的客户端和记录子分段，而不会出现问题。但是，如果你尝试在启动代码中使用经过检测的客户端，你会得SegmentNotFoundException到.

启动代码在 Web 应用程序的标准请求/响应流之外运行，因此您需要手动创建分段以进行检测。Scorekeep 在其 `WebConfig` 文件中显示启动代码的检测。Scorekeep 在启动期间调用 SQL 数据库和 Amazon SNS。



默认 `WebConfig` 类创建通知的 Amazon SNS 订阅。为了提供 X-Ray 开发工具包在使用 Amazon SNS; 客户端时写入的分段，Scorekeep 将在全局记录器上调用 `beginSegment` 和 `endSegment`。

Example **`src/main/java/scorekeep/WebConfig.java`** - 启动代码中的检测过的 AWS 开发工具包客户端

```
AWSXRay.beginSegment("Scorekeep-init");
```

```
if ( System.getenv("NOTIFICATION_EMAIL") != null ){
  try { Sns.createSubscription(); }
  catch (Exception e ) {
    logger.warn("Failed to create subscription for email "+
 System.getenv("NOTIFICATION_EMAIL"));
  }
}
AWSXRay.endSegment();
```

在连接 Amazon RDS 数据库时 Scorekeep 使用的 `RdsWebConfig` 中，配置还为 Hibernate 在启动期间应用数据库架构时使用的 SQL 客户端创建一个分段。

Example **src/main/java/scorekeep/RdsWebConfig.java** - 启动代码中的检测过的 SQL 数据库客户端

```
@PostConstruct
public void schemaExport() {
  EntityManagerFactoryImpl entityManagerFactoryImpl = (EntityManagerFactoryImpl)
 localContainerEntityManagerFactoryBean.getNativeEntityManagerFactory();
  SessionFactoryImplementor sessionFactoryImplementor =
 entityManagerFactoryImpl.getSessionFactory();
  StandardServiceRegistry standardServiceRegistry =
 sessionFactoryImplementor.getSessionFactoryOptions().getServiceRegistry();
  MetadataSources metadataSources = new MetadataSources(new
 BootstrapServiceRegistryBuilder().build());
  metadataSources.addAnnotatedClass(GameHistory.class);
  MetadataImplementor metadataImplementor = (MetadataImplementor)
 metadataSources.buildMetadata(standardServiceRegistry);
  SchemaExport schemaExport = new SchemaExport(standardServiceRegistry,
 metadataImplementor);

  AWSXRay.beginSegment("Scorekeep-init");
  schemaExport.create(true, true);
  AWSXRay.endSegment();
}
```

`SchemaExport` 自动运行并使用 SQL 客户端。由于对客户端进行检测，Scorekeep 必须覆盖默认实现并提供在调用客户端时开发工具包要使用的分段。

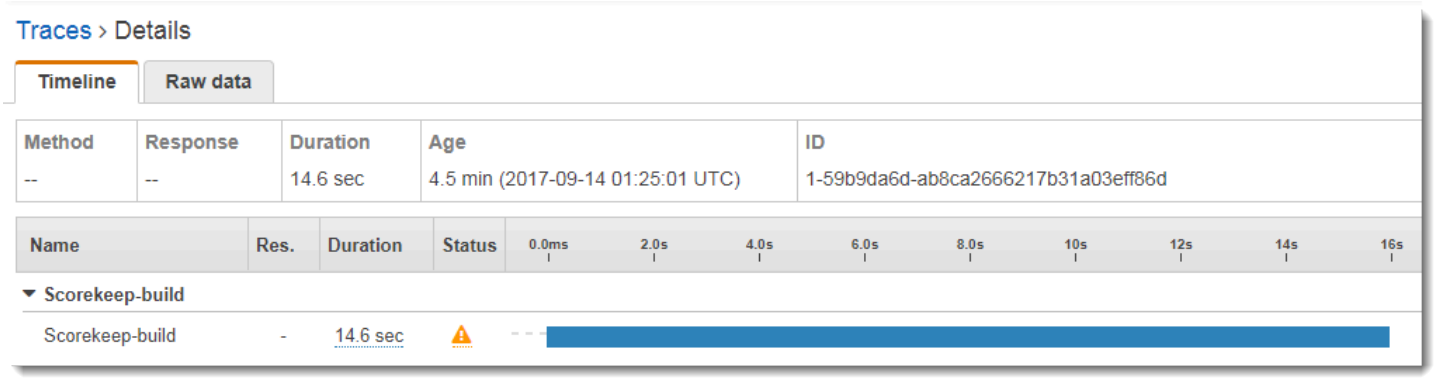
检测脚本

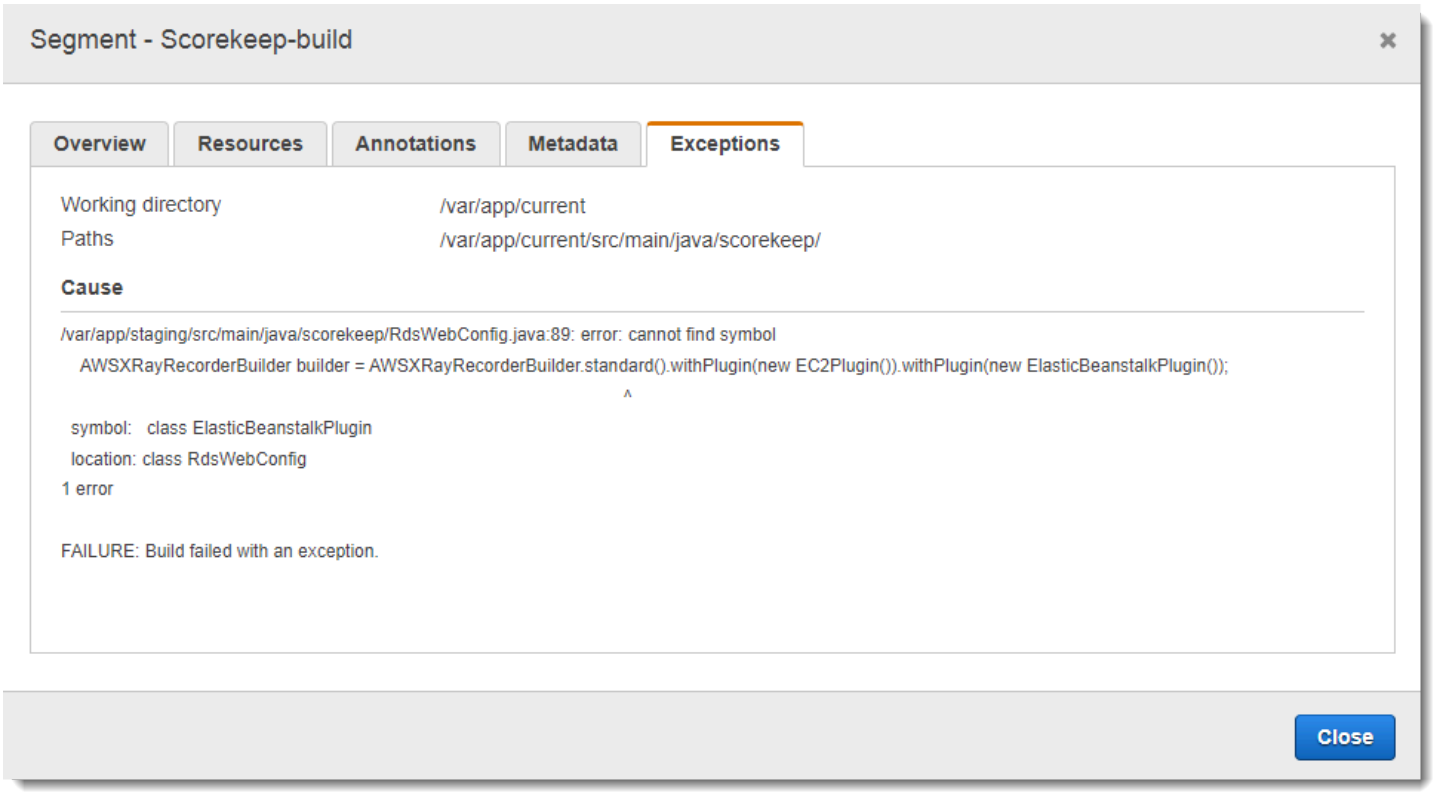
您还可以检测不属于您的应用程序的代码。当 X-Ray 进程守护程序正在运行时，它会将收到的任何分段中继到 X-Ray，即使它们不是由 X-Ray SDK 生成的。Scorekeep 使用自己的脚本来检测用于在部署过程中编译应用程序的构建方式。

Example **bin/build.sh** - 检测过的生成脚本

```
SEGMENT=$(python bin/xray_start.py)
gradle build --quiet --stacktrace &> /var/log/gradle.log; GRADLE_RETURN=$?
if (( GRADLE_RETURN != 0 )); then
  echo "Gradle failed with exit status $GRADLE_RETURN" >&2
  python bin/xray_error.py "$SEGMENT" "$(cat /var/log/gradle.log)"
  exit 1
fi
python bin/xray_success.py "$SEGMENT"
```

xray_start.py、xray_error.py 和 xray_success.py 是简单的 Python 脚本，用于构建分段对象，将它们转换为 JSON 文档并将其通过 UDP 发送到进程守护程序。如果 Gradle 构建失败，您可以通过单击 X-Ray 控制台跟踪地图中的 scorekeep-build 节点，找到错误消息。





检测 Web 应用程序客户端

在 `xray-cognito` 分支中，Scorekeep 使用 Amazon Cognito 使用户能够创建账户并使用该账户登录，以便从 Amazon Cognito 用户池中检索各自的用户信息。当用户登录时，Scorekeep 会使用 Amazon Cognito 身份池来获取用于的 AWS 临时证书。 适用于 JavaScript 的 AWS SDK

身份池配置为允许已登录用户将跟踪数据写入到 AWS X-Ray。Web 应用程序使用这些凭证来记录已登录用户的 ID、浏览器路径以及从客户端角度对 Scorekeep API 的调用。

大多数工作在名为 `xray` 的服务类中完成。此服务类提供了方法来生成必需的标识符、创建进行中的分段，对分段进行最终处理以及将分段文档发送给 X-Ray API。

Example **`public/xray.js`** - 记录和上传分段

```
...
  service.beginSegment = function() {
    var segment = {};
    var traceId = '1-' + service.getHexTime() + '-' + service.getHexId(24);

    var id = service.getHexId(16);
    var startTime = service.getEpochTime();
```

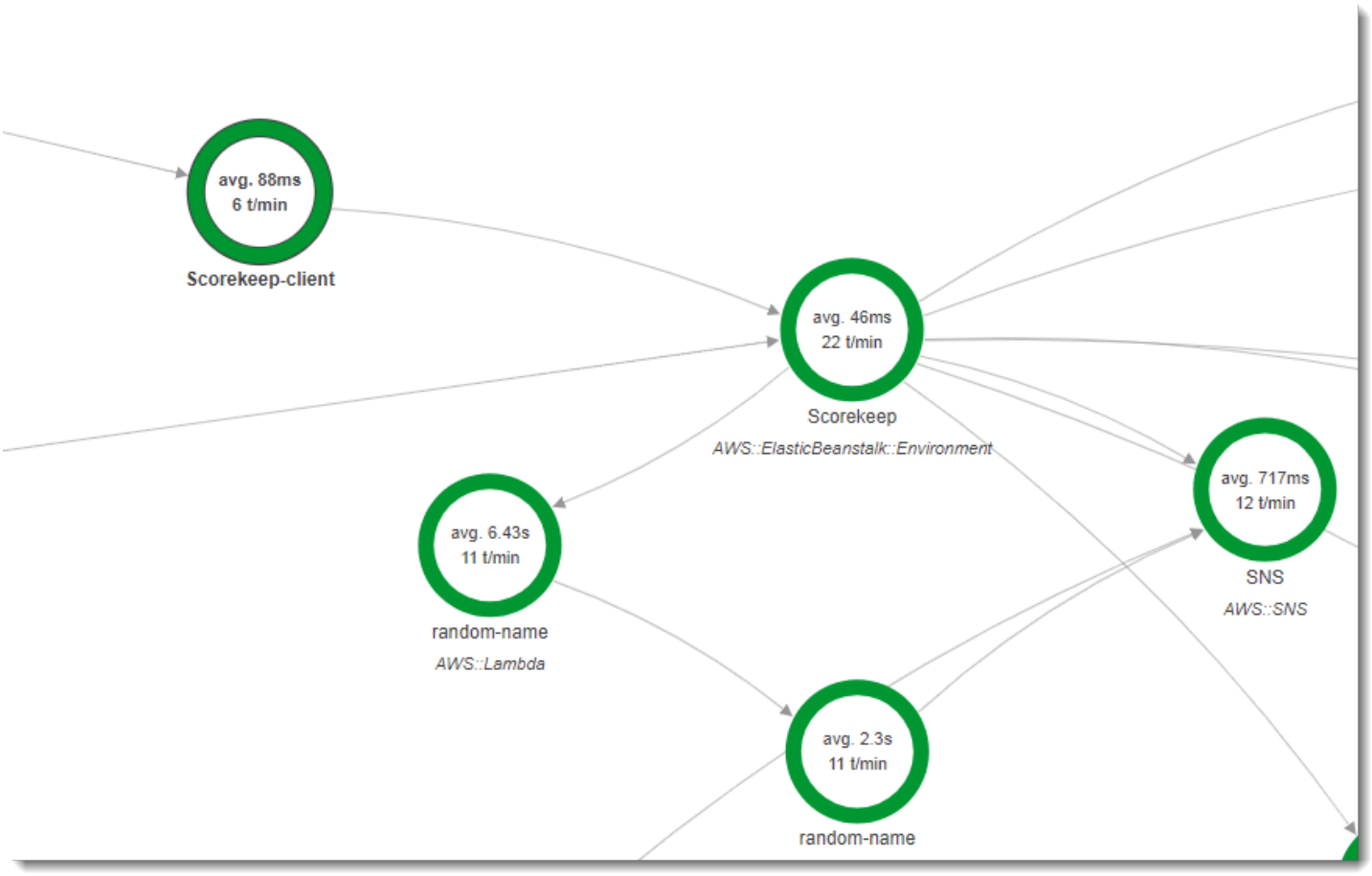
```
  segment.trace_id = traceId;
  segment.id = id;
  segment.start_time = startTime;
  segment.name = 'Scorekeep-client';
  segment.in_progress = true;
  segment.user =  sessionStorage['userid'];
  segment.http = {
    request: {
      url: window.location.href
    }
  };

  var documents = [];
  documents[0] = JSON.stringify(segment);
  service.putDocuments(documents);
  return segment;
}

service.endSegment = function(segment) {
  var endTime = service.getEpochTime();
  segment.end_time = endTime;
  segment.in_progress = false;
  var documents = [];
  documents[0] = JSON.stringify(segment);
  service.putDocuments(documents);
}

service.putDocuments = function(documents) {
  var xray = new AWS.XRay();
  var params = {
    TraceSegmentDocuments: documents
  };
  xray.putTraceSegments(params, function(err, data) {
    if (err) {
      console.log(err, err.stack);
    } else {
      console.log(data);
    }
  })
}
```

这些方法在 Web 应用程序用来调用 Scorekeep API 的资源服务的标头和 `transformResponse` 函数中调用。要将客户端分段与 API 生成的分段包括在同一跟踪中，Web 应用程序必须在 X-Ray SDK 可读取的跟踪标头（`X-Amzn-Trace-Id`）中包含跟踪 ID 和分段 ID。当检测的 Java 应用程序收到包含此标头的请求时，X-Ray SDK for Java 使用相同的跟踪 ID，并使来自 Web 应用程序客户端的分段成为其分段的父分段。

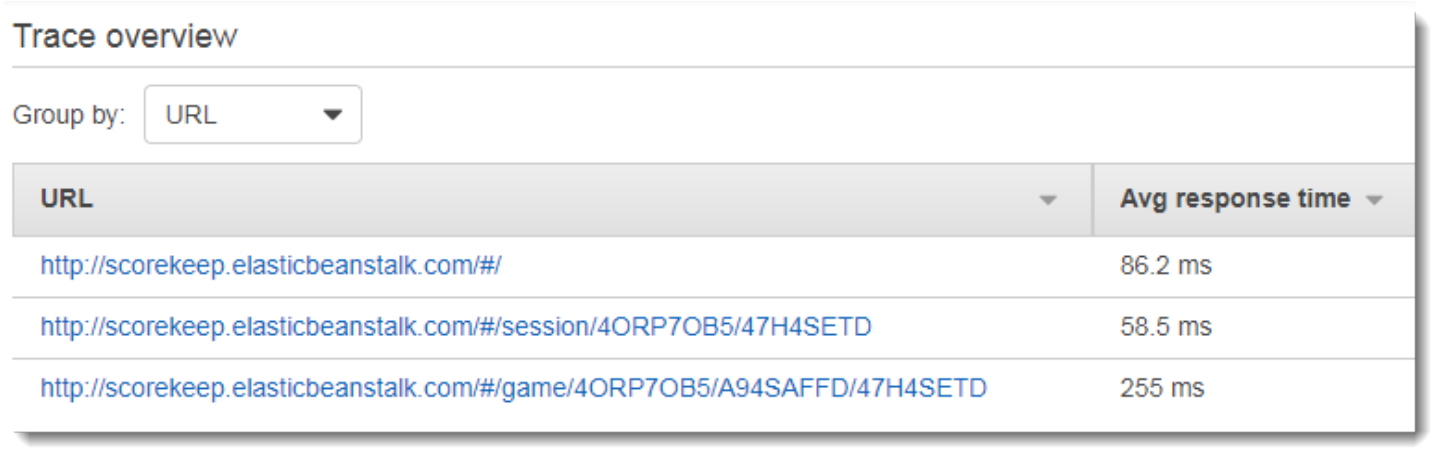
Example **public/app/services.js** - 记录 Angular 资源调用分段和编写跟踪标头

```
var module = angular.module('scorekeep');
module.factory('SessionService', function($resource, api, XRay) {
  return $resource(api + 'session/:id', { id: '@_id' }, {
    segment: {},
    get: {
      method: 'GET',
      headers: {
        'X-Amzn-Trace-Id': function(config) {
          segment = XRay.beginSegment();
          return XRay.getTraceHeader(segment);
        }
      },
      transformResponse: function(data) {
        XRay.endSegment(segment);
        return angular.fromJson(data);
      },
    },
...
```

生成的跟踪地图包含 Web 应用程序客户端的节点。



包含来自 Web 应用程序的分段的跟踪显示用户在浏览器中可见的 URL（以 `/#/` 开头的路径）。如果没有客户端检测，您只会获取 Web 应用程序调用的 API 资源的 URL（以 `/api/` 开头的路径）。



在工作线程中使用检测的客户端

当用户在游戏中获胜后，Scorekeep 使用工作线程向 Amazon SNS 发布通知。发布通知的时间会比请求操作其余部分的总时间更长，并且不会影响客户端或用户。因此，以异步方式执行任务是一种改进响应时间的好方法。

但是，在创建线程时，适用于 Java 的 X-Ray 开发工具包不知道哪个分段处于活动状态。结果，当你尝试在线程中使用经过检测的 适用于 Java 的 AWS SDK 客户端时，它会抛出，从而使线程崩溃。`SegmentNotFoundException`

Example Web-1.error.log

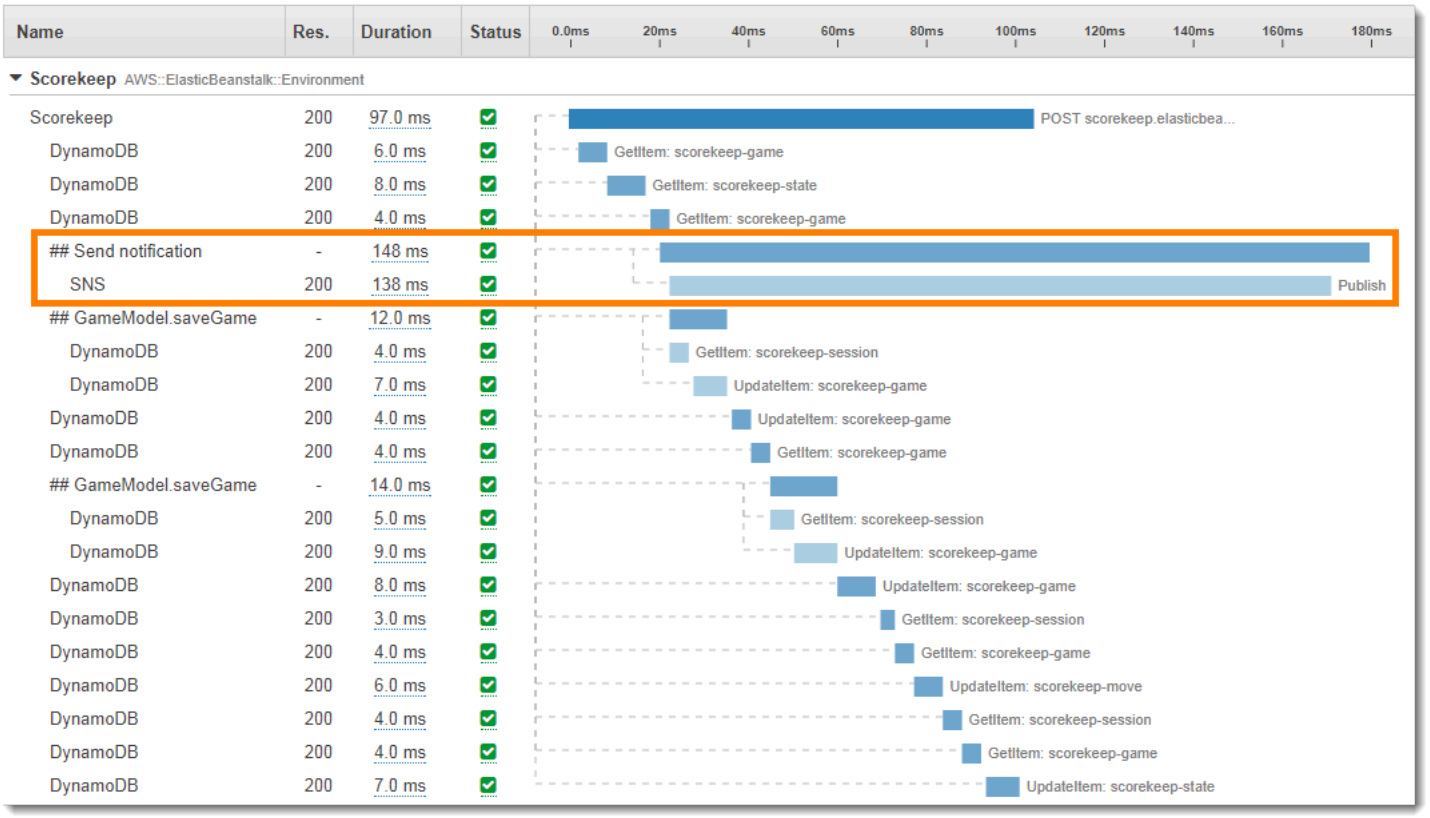
```
Exception in thread "Thread-2" com.amazonaws.xray.exceptions.SegmentNotFoundException:
 Failed to begin subsegment named 'AmazonSNS': segment cannot be found.
        at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
        at
 sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:62)
        at
 sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.ja
...
```

为了解决这个问题，应用程序使用 `GetTraceEntity` 来获取对主线程中的分段的引用，并获取 `Entity.run()` 以安全地运行包含对该分段的上下文具有访问权限的工作线程代码。

Example **src/main/java/scorekeep/MoveFactory.java** - 将跟踪上下文传递到工作线程

```
import com.amazonaws.xray.AWSXRay;
import com.amazonaws.xray.AWSXRayRecorder;
import com.amazonaws.xray.entities.Entity;
import com.amazonaws.xray.entities.Segment;
import com.amazonaws.xray.entities.Subsegment;
...
      Entity segment = recorder.getTraceEntity();
      Thread comm = new Thread() {
        public void run() {
         segment.run(() -> {
            Subsegment subsegment = AWSXRay.beginSubsegment("## Send notification");
            Sns.sendNotification("Scorekeep game completed", "Winner: " + userId);
            AWSXRay.endSubsegment();
          }
        }
```

现在，由于请求在对 Amazon SNS 的调用前已解析，应用程序会为线程创建一个单独的子分段。这可以防止 X-Ray 开发工具包在记录来自 Amazon SNS 的响应之前关闭分段。如果在 Scorekeep 解析请求时未打开任何子分段，来自 Amazon SNS 的响应可能会丢失。



有关多线程处理的更多信息，请参阅[在多线程应用程序中的线程之间传递分段上下文](#)。

AWS X-Ray 守护程序

> **Note**
>
> 现在，您可以使用 CloudWatch 代理从 Amazon EC2 实例和本地服务器收集指标、日志和跟踪。 CloudWatch 代理版本 1.300025.0 及更高版本可以从我们的 X-Ray 客户端收集痕迹 SDKs，然后将其发送到 OpenTelemetryX-R ay。使用 CloudWatch 代理代替 AWS Distro for OpenTelemetry (ADOT) Collector 或 X-Ray 守护程序来收集跟踪可以帮助您减少管理的代理数量。有关更多信息，请参阅《 CloudWatch 用户指南》中的CloudWatch 代理主题。

AWS X-Ray 守护程序是一个软件应用程序，它监听 UDP 端口 2000 上的流量，收集原始数据段数据并将其中继到 API。 AWS X-Ray 该守护程序与配合使用， AWS X-Ray SDKs 并且必须处于运行状态，这样发送的数据 SDKs 才能到达 X-Ray 服务。X-Ray 进程守护程序是一个开源项目。你可以关注该项目并在 github 上 GitHub提交议题和拉取请求。 com/aws/aws-xray-daemon

在 and 上 AWS Lambda AWS Elastic Beanstalk，使用这些服务与 X-Ray 的集成来运行守护程序。每次对采样请求调用函数时，Lambda 都会自动运行该进程守护程序。在 Elastic Beanstalk 上，使用 `XRayEnabled` 配置选项在您环境中的实例上运行该进程守护程序。有关更多信息，请参阅

要在本地、本地或其他地方运行 X-Ray 守护程序 AWS 服务，请下载并运行它，然后授予其将分段文档上传到 X-Ray 的权限。

下载进程守护程序

你可以从 Amazon S3、Amazon ECR 或 Docker Hub 下载守护程序，然后在本地运行它，或者在启动时将其安装在亚马逊 EC2 实例上。

Amazon S3

X-Ray 进程守护程序安装程序和可执行文件

- Linux（可执行文件）- `aws-xray-daemon-linux-3.x.zip`（sig）
- Linux（RPM 安装程序）– `aws-xray-daemon-3.x.rpm`
- Linux（DEB 安装程序）– `aws-xray-daemon-3.x.deb`
- Linux（ARM64，可执行文件）— `aws-xray-daemon-linux-arm64-3.x.zip`（签名）

- Linux（ARM64，RPM 安装程序）— aws-xray-daemon-arm64-3.x.rpm
- Linux（ARM64，DEB 安装程序）— **aws-xray-daemon-arm64-3.x.deb**
- OS X（可执行文件）- aws-xray-daemon-macos-3.x.zip（sig）
- Windows（可执行文件）– aws-xray-daemon-windows-process-3.x.zip（sig）
- Windows（服务）– aws-xray-daemon-windows-service-3.x.zip（sig）

这些链接始终指向最新 3.x 版本的进程守护程序。要下载特定版本，请执行以下操作：

- 如果要下载 3.3.0 之前的版本，请将 3.x 替换为该版本号。例如，2.1.0。在 3.3.0 版本之前，唯一可用的架构是 arm64。例如，2.1.0 和 arm64。
- 如果要下载 3.3.0 之后的版本，请将 3.x 替换为该版本号，将 arch 替换为该架构类型。

X-Ray 资产被复制到每个受支持区域的存储桶。要使用离您最近的存储桶或您的 AWS 资源，请将上述链接中的区域替换为您的区域。

```
https://s3.us-west-2.amazonaws.com/aws-xray-assets.us-west-2/xray-daemon/aws-xray-daemon-3.x.rpm
```

Amazon ECR

从 3.2.0 版本开始，可以在 Amazon ECR 找到该进程守护程序。在提取映像前，应该先对连接 Amazon ECR 公共注册表的 Docker 客户端进行身份验证。

运行以下命令提取最新发布的 3.x 版本标签：

```
docker pull public.ecr.aws/xray/aws-xray-daemon:3.x
```

可通过将 3.x 替换为 alpha 或是某一具体版本号来下载以前的版本或 Alpha 版本。不建议在生产环境中使用带有 Alpha 标签的进程守护程序标签。

Docker Hub

可以在 Docker Hub 上找到该进程守护程序。运行以下命令下载最新发布的 3.x 版本：

```
docker pull amazon/aws-xray-daemon:3.x
```

通过将 3.x 替换为想要的版本，可以发布以前版本的进程守护程序。

验证进程守护程序存档的签名

以 ZIP 存档格式压缩的进程守护程序资产会附带 GPG 签名文件。公有密钥在这里：aws-xray.gpg。

您可以使用公有密钥来验证进程守护程序的 ZIP 存档是原始的且未经过修改。首先，使用 GnuPG 导入公有密钥。

导入公有密钥

1. 下载公有密钥。

```
$ BUCKETURL=https://s3.us-east-2.amazonaws.com/aws-xray-assets.us-east-2
$ wget $BUCKETURL/xray-daemon/aws-xray.gpg
```

2. 将公有密钥导入到您的密钥环中。

```
$ gpg --import aws-xray.gpg
gpg: /Users/me/.gnupg/trustdb.gpg: trustdb created
gpg: key 7BFE036BFE6157D3: public key "AWS X-Ray <aws-xray@amazon.com>" imported
gpg: Total number processed: 1
gpg:               imported: 1
```

使用导入的密钥来验证进程守护程序 ZIP 存档的签名。

验证存档的签名

1. 下载存档和签名文件。

```
$ BUCKETURL=https://s3.us-east-2.amazonaws.com/aws-xray-assets.us-east-2
$ wget $BUCKETURL/xray-daemon/aws-xray-daemon-linux-3.x.zip
$ wget $BUCKETURL/xray-daemon/aws-xray-daemon-linux-3.x.zip.sig
```

2. 运行 `gpg --verify` 来验证签名。

```
$ gpg --verify aws-xray-daemon-linux-3.x.zip.sig aws-xray-daemon-linux-3.x.zip
gpg: Signature made Wed 19 Apr 2017 05:06:31 AM UTC using RSA key ID FE6157D3
gpg: Good signature from "AWS X-Ray <aws-xray@amazon.com>"
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: EA6D 9271 FBF3 6990 277F  4B87 7BFE 036B FE61 57D3
```

请注意有关信任的警告。只有当您或您信任的某个人对文件进行了签名，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。

运行进程守护程序

在本地从命令行运行进程守护程序。使用 -o 选项以本地模式运行，-n 选项设置区域。

```
~/Downloads$ ./xray -o -n us-east-2
```

有关特定于平台的详细说明，请参阅下列主题：

- Linux（本地）- 在 Linux 上运行 X-Ray 进程守护程序
- Windows（本地）- 在 Windows 上运行 X-Ray 进程守护程序
- Elastic Beanstalk – 正在运行 X-Ray 守护程序 AWS Elastic Beanstalk
- 亚马逊 EC2 — 在亚马逊上运行 X-Ray 守护程序 EC2
- Amazon ECS – 在 Amazon ECS 上运行 X-Ray 进程守护程序

您可以使用命令行选项或配置文件进一步自定义进程守护程序的行为。有关详细信息，请参阅配置 AWS X-Ray 守护程序。

授予进程守护程序向 X-Ray 发送数据的权限

X-Ray 守护程序使用 AWS SDK 将跟踪数据上传到 X-Ray，它需要有权限的 AWS 凭证才能这样做。

在 Amazon 上 EC2，守护程序会自动使用实例的实例配置文件角色。有关在本地运行该进程守护程序所需凭证的相关信息，请参阅在本地运行应用程序。

如果您在多个位置（凭证文件、实例配置文件或环境变量）指定凭证，SDK 会提供证书链，决定使用哪个凭证。有关向 SDK 提供凭证的更多信息，请参阅《AWS SDK for Go 开发人员指南》中的指定凭证。

进程守护程序的凭证所属的 IAM 角色或用户必须有权代表您写入数据到服务。

- 要在 Amazon 上使用守护程序 EC2，请创建一个新的实例配置文件角色或将托管策略添加到现有角色中。

- 要在 Elastic Beanstalk 上使用进程守护程序，请将托管策略添加到 Elastic Beanstalk 默认实例配置文件角色。
- 请参阅在本地运行应用程序，了解如何在本地运行进程守护程序。

有关更多信息，请参阅 的身份和访问管理 AWS X-Ray。

X-Ray 进程守护程序日志

守护程序输出有关其当前配置和发送到 AWS X-Ray的段的信息。

```
2016-11-24T06:07:06Z [Info] Initializing AWS X-Ray daemon 2.1.0
2016-11-24T06:07:06Z [Info] Using memory limit of 49 MB
2016-11-24T06:07:06Z [Info] 313 segment buffers allocated
2016-11-24T06:07:08Z [Info] Successfully sent batch of 1 segments (0.123 seconds)
2016-11-24T06:07:09Z [Info] Successfully sent batch of 1 segments (0.006 seconds)
```

默认情况下，进程守护程序将日志输出到 STDOUT。如果您在后台运行进程守护程序，请使用 `--log-file` 命令行选项或配置文件来设置日志文件的路径。您也可以设置日志级别并禁用日志轮换。有关说明，请参阅配置 AWS X-Ray 守护程序：

在 Elastic Beanstalk 中，平台将设置进程守护程序日志的位置。有关详细信息，请参阅正在运行 X-Ray 守护程序 AWS Elastic Beanstalk。

配置 AWS X-Ray 守护程序

您可以使用命令行选项或配置文件来自定义 X-Ray 进程守护程序的行为。大多数选项适用于这两种方法，但有一些只适用于配置文件，有一些只适用于命令行。

要开始使用，您需要知道的唯一选项是 `-n` 或 `--region`，用于设置进程守护程序使用的区域，或将跟踪数据发送到 X-Ray。

```
~/xray-daemon$ ./xray -n us-east-2
```

如果您在本地运行守护程序，也就是说，不是在 Amazon 上 EC2，则可以添加跳过检查实例配置文件凭证的-o选项，这样守护程序就可以更快地准备就绪。

```
~/xray-daemon$ ./xray -o -n us-east-2
```

其余的命令行选项可让您配置日志记录、侦听不同端口、限制进程守护程序可以使用的内存量，或代入角色将跟踪数据发送到其他账户。

您可以将配置文件传递到进程守护程序，以访问高级配置选项，并完成其他任务，如限制并发调用 X-Ray 的数量、禁用日志轮换，并将流量发送到代理。

Sections

支持的环境变量

X-Ray 进程守护程序支持以下环境变量：

- `AWS_REGION` — 指定 X-Ray 服务端点的 AWS 区域。
- `HTTPS_PROXY` — 为进程守护程序指定一个通过其上传分段的代理地址。它可以是代理服务器使用的 DNS 域名或 IP 地址和端口号。

使用命令行选项

当您在本地或使用用户数据脚本运行进程守护程序时，将这些选项传递给它。

命令行选项

- `-b、--bind` - 侦听其他 UDP 端口上的分段文档。

```
--bind "127.0.0.1:3000"
```

  默认值 – `2000`。
- `-t、--bind-tcp` - 侦听对其他 TCP 端口上的 X-Ray 服务的调用。

```
-bind-tcp "127.0.0.1:3000"
```

  默认值 – `2000`。
- `-c、--config` - 从指定路径加载配置文件。

```
--config "/home/ec2-user/xray-daemon.yaml"
```

- -f、--log-file - 将日志输出到指定文件路径。

```
--log-file "/var/log/xray-daemon.log"
```

- -l、--log-level - 日志级别，依次为从最详细到最不详细：
  dev、debug、info、warn、error、prod。

```
--log-level warn
```

  默认值 – prod

- -m、--buffer-memory - 更改缓冲区可以使用的内存量（单位：MB，最小值为 3）。

```
--buffer-memory 50
```

  默认值 - 1% 的可用内存。

- -o，--local-mode— 不检查 EC2 实例元数据。
- -r、--role-arn - 采用指定的 IAM 角色将分段上传到其他账户。

```
--role-arn "arn:aws:iam::123456789012:role/xray-cross-account"
```

- -a，--resource-arn— 运行守护程序的资源的亚马逊 AWS 资源名称 (ARN)。
- -p，--proxy-address— AWS X-Ray 通过代理将区段上传到。必须指定代理服务器的协议。

```
--proxy-address "http://192.0.2.0:3000"
```

- -n、--region - 向特定区域中的 X-Ray 服务发送分段。
- -v，--version— 显示 AWS X-Ray 守护程序版本。
- -h、--help - 显示帮助屏幕。

使用配置文件

您也可以使用 YAML 格式文件来配置进程守护程序。使用 -c 选项传递配置文件到进程守护程序。

```
~$ ./xray -c ~/xray-daemon.yaml
```

配置文件选项

- TotalBufferSizeMB - 最大缓冲区大小（单位：MB，最小值为 3）。选择 0 则使用 1% 的主机内存。
- Concurrency— 上载分段文档的最大并发调用次数。 AWS X-Ray
- Region— 将区段发送到特定区域的 AWS X-Ray 服务。
- Socket - 配置进程守护程序的绑定。
  - UDPAddress - 更改进程守护程序侦听的端口。
  - TCPAddress - 侦听对其他 TCP 端口上的 X-Ray 服务的调用。
- Logging - 配置日志记录行为。
  - LogRotation - 设置为 false 以禁用日志轮换。
  - LogLevel - 更改日志级别，依次为从最详细到最不详细：dev、debug、info 或 prod、warn、error、prod。默认为 prod，等同于 info。
  - LogPath - 将日志输出到指定文件路径。
- LocalMode— 设置true为可跳过对 EC2 实例元数据的检查。
- ResourceARN— 运行守护程序的资源的亚马逊 AWS 资源名称 (ARN)。
- RoleARN - 采用指定的 IAM 角色将分段上传到其他账户。
- ProxyAddress— AWS X-Ray 通过代理将区段上传到。
- Endpoint - 将 X-Ray 服务端点改为进程守护程序向其发送分段文档的节点。
- NoVerifySSL - 禁用 TLS 证书验证。
- Version - 进程守护程序配置文件格式版本。文件格式版本是必填字段。

Example Xray-daemon.yaml

此配置文件将进程守护程序的侦听端口改为 3000，并且关闭对实例元数据的检查，设置用于上传分段的角色，以及更改区域和日志记录选项。

```
Socket:
  UDPAddress: "127.0.0.1:3000"
  TCPAddress: "127.0.0.1:3000"
Region: "us-west-2"
Logging:
  LogLevel: "warn"
  LogPath: "/var/log/xray-daemon.log"
LocalMode: true
```

```
RoleARN: "arn:aws:iam::123456789012:role/xray-cross-account"
Version: 2
```

在本地运行 X-Ray 进程守护程序

你可以在 Linux、macOS、Windows 或 Docker 容器中本地运行 AWS X-Ray 守护程序。当您在开发和测试经过检测的应用程序时，运行进程守护程序可将跟踪数据中继到 X-Ray。使用此处的说明下载并解压缩进程守护程序。

在本地运行时，守护程序可以从 AWS SDK 凭据文件（`.aws/credentials`在您的用户目录中）或环境变量中读取凭据。有关更多信息，请参阅 授予进程守护程序向 X-Ray 发送数据的权限。

进程守护程序在端口 2000 上侦听 UDP 数据。您可以使用配置文件和命令行选项更改端口和其他选项。有关更多信息，请参阅 配置 AWS X-Ray 守护程序。

在 Linux 上运行 X-Ray 进程守护程序

您可以从命令行运行进程守护程序可执行文件。使用 `-o` 选项以本地模式运行，`-n` 选项设置区域。

```
~/xray-daemon$ ./xray -o -n us-east-2
```

要在后台运行进程守护程序，请使用 &。

```
~/xray-daemon$ ./xray -o -n us-east-2 &
```

使用 `pkill` 终止在后台运行的进程守护程序进程。

```
~$ pkill xray
```

在 Docker 容器中运行 X-Ray 进程守护程序

要在 Docker 容器中本地运行进程守护程序，请将以下文本保存为名为 `Dockerfile` 的文件。在 Amazon ECR 上下载完整的示例映像。有关更多信息，请参阅下载进程守护程序。

Example Dockerfile — Amazon Linux

```
FROM amazonlinux
RUN yum install -y unzip
```

```
RUN curl -o daemon.zip https://s3.us-east-2.amazonaws.com/aws-xray-assets.us-east-2/
xray-daemon/aws-xray-daemon-linux-3.x.zip
RUN unzip daemon.zip && cp xray /usr/bin/xray
ENTRYPOINT ["/usr/bin/xray", "-t", "0.0.0.0:2000", "-b", "0.0.0.0:2000"]
EXPOSE 2000/udp
EXPOSE 2000/tcp
```

利用 `docker build` 构建容器映像。

```
~/xray-daemon$ docker build -t xray-daemon .
```

利用 `docker run` 在容器中运行映像。

```
~/xray-daemon$ docker run \
      --attach STDOUT \
      -v ~/.aws/:/root/.aws/:ro \
      --net=host \
      -e AWS_REGION=us-east-2 \
      --name xray-daemon \
      -p 2000:2000/udp \
      xray-daemon -o
```

此命令使用以下选项：

- `--attach STDOUT` - 在终端查看进程守护程序的输出。
- `-v ~/.aws/:/root/.aws/:ro`— 授予容器对`.aws`目录的只读访问权限，使其能够读取您的 AWS SDK 凭据。
- `AWS_REGION=`*`us-east-2`* - 设置 `AWS_REGION` 环境变量，以通知进程守护程序使用哪个区域。
- `--net=host` - 将容器附加到 `host` 网络。主机网络上的容器无需发布端口即可互相通信。
- `-p 2000:2000/udp` - 将您计算机上的 UDP 端口 2000 映射到容器上的同一端口。对于同一网络中的容器进行通信而言，这不是必需的，但它允许您通过命令行或通过未在 Docker 中运行的应用程序将分段发送到进程守护程序。
- `--name xray-daemon` - 命名容器 `xray-daemon`，而不是随机生成名称。
- `-o`（位于映像名称后）- 将 `-o` 选项追加到入口点（在容器中运行进程守护程序）。此选项告诉守护程序在本地模式下运行，以防止它尝试读取 Amazon EC2 实例元数据。

要停止进程守护程序，请使用 `docker stop`。如果您更改 `Dockerfile` 并生成新映像，需要删除现有容器，然后才能创建另一个具有相同名称的容器。使用 `docker rm` 删除容器。

```
$ docker stop xray-daemon
$ docker rm xray-daemon
```

在 Windows 上运行 X-Ray 进程守护程序

您可以从命令行运行进程守护程序可执行文件。使用 -o 选项以本地模式运行，-n 选项设置区域。

```
> .\xray_windows.exe -o -n us-east-2
```

使用 PowerShell 脚本为守护程序创建和运行服务。

Example PowerShell 脚本-Windows

```
if ( Get-Service "AWSXRayDaemon" -ErrorAction SilentlyContinue ){
    sc.exe stop AWSXRayDaemon
    sc.exe delete AWSXRayDaemon
}
if ( Get-Item -path aws-xray-daemon -ErrorAction SilentlyContinue ) {
    Remove-Item -Recurse -Force aws-xray-daemon
}

$currentLocation = Get-Location
$zipFileName = "aws-xray-daemon-windows-service-3.x.zip"
$zipPath = "$currentLocation\$zipFileName"
$destPath = "$currentLocation\aws-xray-daemon"
$daemonPath = "$destPath\xray.exe"
$daemonLogPath = "C:\inetpub\wwwroot\xray-daemon.log"
$url = "https://s3.dualstack.us-west-2.amazonaws.com/aws-xray-assets.us-west-2/xray-
daemon/aws-xray-daemon-windows-service-3.x.zip"

Invoke-WebRequest -Uri $url -OutFile $zipPath
Add-Type -Assembly "System.IO.Compression.Filesystem"
[io.compression.zipfile]::ExtractToDirectory($zipPath, $destPath)

sc.exe create AWSXRayDaemon binPath= "$daemonPath -f $daemonLogPath"
sc.exe start AWSXRayDaemon
```

在 OS X 上运行 X-Ray 进程守护程序

您可以从命令行运行进程守护程序可执行文件。使用 -o 选项以本地模式运行，-n 选项设置区域。

```
~/xray-daemon$ ./xray_mac -o -n us-east-2
```

要在后台运行进程守护程序，请使用 &。

```
~/xray-daemon$ ./xray_mac -o -n us-east-2 &
```

使用 nohup 可防止在终端关闭时终止进程守护程序。

```
~/xray-daemon$ nohup ./xray_mac &
```

正在运行 X-Ray 守护程序 AWS Elastic Beanstalk

要将应用程序中的跟踪数据中继到 AWS X-Ray，您可以在 Elastic Beanstalk 环境的亚马逊实例上运行 X-Ray 守护程序。 EC2 有关受支持平台的列表，请参阅 AWS Elastic Beanstalk 开发人员指南中的配置 AWS X-Ray 调试。

> **Note**
>
> 该进程守护程序使用环境的实例配置文件获取权限。有关将权限添加到 Elastic Beanstalk 实例配置文件的说明，请参阅授予进程守护程序向 X-Ray 发送数据的权限。

Elastic Beanstalk 平台提供配置选项，您可以设置它，自动运行进程守护程序。您可以在源代码的配置文件中启用进程守护程序，或者通过在 Elastic Beanstalk 控制台中选择选项来启用。启用配置选项后，进程守护程序将安装到实例上，并作为服务运行。

Elastic Beanstalk 平台上包括的版本可能不是最新版本。请参阅支持的平台主题，找出您的平台配置适用的进程守护程序版本。

Elastic Beanstalk 在多容器 Docker (Amazon ECS) 平台上不提供 X-Ray 进程守护程序。

使用 Elastic Beanstalk X-Ray 集成运行 X-Ray 进程守护程序

使用控制台启用 X-Ray 集成，或者在应用程序源代码中使用配置文件来配置。

在 Elastic Beanstalk 控制台中启用 X-Ray 进程守护程序

1. 打开 Elastic Beanstalk 控制台。

2. 导航到您的环境的管理控制台。
3. 选择配置。
4. 选择软件设置。
5. 对于 X-Ray 进程守护程序，选择已启动。
6. 选择应用。

您可以在源代码中包含配置文件，使得您的配置可以在环境之间移植。

Example .ebextensions/xray-daemon.config

```
option_settings:
  aws:elasticbeanstalk:xray:
    XRayEnabled: true
```

Elastic Beanstalk 将配置文件传递到进程守护程序并将日志输出到标准位置。

在 Windows Server 平台上

- 配置文件 - `C:\Program Files\Amazon\XRay\cfg.yaml`
- 日志 - `c:\Program Files\Amazon\XRay\logs\xray-service.log`

在 Linux 平台上

- 配置文件 - `/etc/amazon/xray/cfg.yaml`
- 日志 - `/var/log/xray/xray.log`

Elastic Beanstalk 提供了用于从或命令行提取实例日志 AWS Management Console 的工具。您可以使用配置文件添加一项任务，来指示 Elastic Beanstalk 包含 X-Ray 进程守护程序日志。

Example .ebextensions/xray-logs.config - Linux

```
files:
  "/opt/elasticbeanstalk/tasks/taillogs.d/xray-daemon.conf" :
    mode: "000644"
    owner: root
    group: root
    content: |
```

```
      /var/log/xray/xray.log
```

Example .ebextensions/xray-logs.config - Windows Server

```
files:
  "c:/Program Files/Amazon/ElasticBeanstalk/config/taillogs.d/xray-daemon.conf" :
    mode: "000644"
    owner: root
    group: root
    content: |
      c:\Progam Files\Amazon\XRay\logs\xray-service.log
```

有关更多信息，请参阅开发人员指南中的 Elastic Beanstalk 环境 EC2 的 Amazon AWS Elastic Beanstalk 实例查看日志。

手动下载和运行 X-Ray 进程守护程序（高级）

如果 X-Ray 进程守护程序对您的平台配置不可用，则可以从 Amazon S3 下载它并使用配置文件来运行。

使用 Elastic Beanstalk 配置文件下载并运行进程守护程序。

Example .ebextensions/xray.config - Linux

```
commands:
  01-stop-tracing:
    command: yum remove -y xray
    ignoreErrors: true
  02-copy-tracing:
    command: curl https://s3.us-east-2.amazonaws.com/aws-xray-assets.us-east-2/xray-
daemon/aws-xray-daemon-3.x.rpm -o /home/ec2-user/xray.rpm
  03-start-tracing:
    command: yum install -y /home/ec2-user/xray.rpm

files:
  "/opt/elasticbeanstalk/tasks/taillogs.d/xray-daemon.conf" :
    mode: "000644"
    owner: root
    group: root
    content: |
      /var/log/xray/xray.log
  "/etc/amazon/xray/cfg.yaml" :
```

```
    mode: "000644"
    owner: root
    group: root
    content: |
      Logging:
        LogLevel: "debug"
      Version: 2
```

Example .ebextensions/xray.config - Windows Server

```
container_commands:
  01-execute-config-script:
    command: Powershell.exe -ExecutionPolicy Bypass -File c:\\temp\\installDaemon.ps1
    waitAfterCompletion: 0

files:
  "c:/temp/installDaemon.ps1":
    content: |
      if ( Get-Service "AWSXRayDaemon" -ErrorAction SilentlyContinue ) {
          sc.exe stop AWSXRayDaemon
          sc.exe delete AWSXRayDaemon
      }

      $targetLocation = "C:\Program Files\Amazon\XRay"
      if ((Test-Path $targetLocation) -eq 0) {
          mkdir $targetLocation
      }

      $zipFileName = "aws-xray-daemon-windows-service-3.x.zip"
      $zipPath = "$targetLocation\$zipFileName"
      $destPath = "$targetLocation\aws-xray-daemon"
      if ((Test-Path $destPath) -eq 1) {
          Remove-Item -Recurse -Force $destPath
      }

      $daemonPath = "$destPath\xray.exe"
      $daemonLogPath = "$targetLocation\xray-daemon.log"
      $url = "https://s3.dualstack.us-west-2.amazonaws.com/aws-xray-assets.us-west-2/
xray-daemon/aws-xray-daemon-windows-service-3.x.zip"

      Invoke-WebRequest -Uri $url -OutFile $zipPath
      Add-Type -Assembly "System.IO.Compression.Filesystem"
      [io.compression.zipfile]::ExtractToDirectory($zipPath, $destPath)
```

```
      New-Service -Name "AWSXRayDaemon" -StartupType Automatic -BinaryPathName
 "`"$daemonPath`" -f `"$daemonLogPath`""
      sc.exe start AWSXRayDaemon
    encoding: plain
  "c:/Program Files/Amazon/ElasticBeanstalk/config/taillogs.d/xray-daemon.conf" :
    mode: "000644"
    owner: root
    group: root
    content: |
      C:\Program Files\Amazon\XRay\xray-daemon.log
```

这些示例还将进程守护程序日志文件添加到了 Elastic Beanstalk 的尾日志任务，以在您通过控制台或 Elastic Beanstalk 命令行界面 (EB CLI) 请求日志时将其包含在内。

在亚马逊上运行 X-Ray 守护程序 EC2

您可以在亚马逊 EC2的以下操作系统上运行 X-Ray 守护程序：

- Amazon Linux
- Ubuntu
- Windows Server（2012 R2 及更高版本）

使用实例配置文件授予进程守护程序权限以上传跟踪数据到 X-Ray。有关更多信息，请参阅 授予进程守护程序向 X-Ray 发送数据的权限。

使用用户数据脚本在启动实例时自动运行进程守护程序。

Example 用户数据脚本 - Linux

```
#!/bin/bash
curl https://s3.us-east-2.amazonaws.com/aws-xray-assets.us-east-2/xray-daemon/aws-xray-
daemon-3.x.rpm -o /home/ec2-user/xray.rpm
yum install -y /home/ec2-user/xray.rpm
```

Example 用户数据脚本 - Windows Server

```
<powershell>
if ( Get-Service "AWSXRayDaemon" -ErrorAction SilentlyContinue ) {
    sc.exe stop AWSXRayDaemon
    sc.exe delete AWSXRayDaemon
```

```
}

$targetLocation = "C:\Program Files\Amazon\XRay"
if ((Test-Path $targetLocation) -eq 0) {
    mkdir $targetLocation
}

$zipFileName = "aws-xray-daemon-windows-service-3.x.zip"
$zipPath = "$targetLocation\$zipFileName"
$destPath = "$targetLocation\aws-xray-daemon"
if ((Test-Path $destPath) -eq 1) {
    Remove-Item -Recurse -Force $destPath
}

$daemonPath = "$destPath\xray.exe"
$daemonLogPath = "$targetLocation\xray-daemon.log"
$url = "https://s3.dualstack.us-west-2.amazonaws.com/aws-xray-assets.us-west-2/xray-
daemon/aws-xray-daemon-windows-service-3.x.zip"

Invoke-WebRequest -Uri $url -OutFile $zipPath
Add-Type -Assembly "System.IO.Compression.Filesystem"
[io.compression.zipfile]::ExtractToDirectory($zipPath, $destPath)

New-Service -Name "AWSXRayDaemon" -StartupType Automatic -BinaryPathName
 "`"$daemonPath`" -f `"$daemonLogPath`""
sc.exe start AWSXRayDaemon
</powershell>
```

在 Amazon ECS 上运行 X-Ray 进程守护程序

在 Amazon ECS 上，创建运行 X-Ray 进程守护程序的 Docker 映像，将其上传到 Docker 映像存储库，然后部署到 Amazon ECS 集群。您可以在任务定义文件中使用端口映射和网络模式设置，允许您的应用程序与进程守护程序容器通信。

使用官方 Docker 映像

X-Ray 在 Amazon ECR 上提供了 Docker 容器映像，您可以与您的应用程序一起部署该映像。有关更多信息，请参阅下载进程守护程序。

Example 任务定义

```
    {
      "name": "xray-daemon",
      "image": "amazon/aws-xray-daemon",
      "cpu": 32,
      "memoryReservation": 256,
      "portMappings" : [
          {
              "hostPort": 0,
              "containerPort": 2000,
              "protocol": "udp"
          }
       ]
    }
```

创建和构建 Docker 映像

对于自定义配置，您可能需要定义自己的 Docker 映像。

为任务角色添加托管式策略，授予进程守护程序将跟踪数据上传到 X-Ray 的权限。有关更多信息，请参阅 授予进程守护程序向 X-Ray 发送数据的权限。

使用以下 Dockerfile 之一来创建运行进程守护程序的映像。

Example Dockerfile — Amazon Linux

```
FROM amazonlinux
RUN yum install -y unzip
RUN curl -o daemon.zip https://s3.us-east-2.amazonaws.com/aws-xray-assets.us-east-2/
xray-daemon/aws-xray-daemon-linux-3.x.zip
RUN unzip daemon.zip && cp xray /usr/bin/xray
ENTRYPOINT ["/usr/bin/xray", "-t", "0.0.0.0:2000", "-b", "0.0.0.0:2000"]
EXPOSE 2000/udp
EXPOSE 2000/tcp
```

> **Note**
>
> 需要 `-t` 和 `-b` 来指定绑定地址侦听多容器环境的环回。

Example Dockerfile - Ubuntu

对于 Debian 衍生物，您还需要安装证书颁发机构 (CA) 证书，以避免下载安装程序时遇到问题。

```
FROM ubuntu:16.04
RUN apt-get update && apt-get install -y --force-yes --no-install-recommends apt-
transport-https curl ca-certificates wget && apt-get clean && apt-get autoremove && rm
 -rf /var/lib/apt/lists/*
RUN wget https://s3.us-east-2.amazonaws.com/aws-xray-assets.us-east-2/xray-daemon/aws-
xray-daemon-3.x.deb
RUN dpkg -i aws-xray-daemon-3.x.deb
ENTRYPOINT ["/usr/bin/xray", "--bind=0.0.0.0:2000", "--bind-tcp=0.0.0.0:2000"]
EXPOSE 2000/udp
EXPOSE 2000/tcp
```

在您的任务定义中，配置取决于您使用的联网模式。桥式联网是默认模式，可在您的默认 VPC 中使用。在桥式网络中，将 AWS_XRAY_DAEMON_ADDRESS 环境变量设置为告诉 X-Ray 开发工具包引用哪个容器端口并设置主机端口。例如，可以发布 UDP 端口 2000，并创建您的应用程序容器与进程守护程序容器之间的链接。

Example 任务定义

```
    {
      "name": "xray-daemon",
      "image": "123456789012.dkr.ecr.us-east-2.amazonaws.com/xray-daemon",
      "cpu": 32,
      "memoryReservation": 256,
      "portMappings" : [
          {
              "hostPort": 0,
              "containerPort": 2000,
              "protocol": "udp"
          }
       ]
    },
    {
      "name": "scorekeep-api",
      "image": "123456789012.dkr.ecr.us-east-2.amazonaws.com/scorekeep-api",
      "cpu": 192,
      "memoryReservation": 512,
      "environment": [
          { "name" : "AWS_REGION", "value" : "us-east-2" },
          { "name" : "NOTIFICATION_TOPIC", "value" : "arn:aws:sns:us-
east-2:123456789012:scorekeep-notifications" },
          { "name" : "AWS_XRAY_DAEMON_ADDRESS", "value" : "xray-daemon:2000" }
```

```
      ],
      "portMappings" : [
          {
              "hostPort": 5000,
              "containerPort": 5000
          }
      ],
      "links": [
        "xray-daemon"
      ]
    }
```

如果您在 VPC 的私有子网中运行集群，可以使用 awsvpc 网络模式将弹性网络接口 (ENI) 附加到您的容器。这样可以避免使用链接。省略端口映射、链接和 AWS_XRAY_DAEMON_ADDRESS 环境变量中的主机端口。

Example VPC 任务定义

```
{
    "family": "scorekeep",
    "networkMode":"awsvpc",
    "containerDefinitions": [
        {
          "name": "xray-daemon",
          "image": "123456789012.dkr.ecr.us-east-2.amazonaws.com/xray-daemon",
          "cpu": 32,
          "memoryReservation": 256,
          "portMappings" : [
              {
                  "containerPort": 2000,
                  "protocol": "udp"
              }
          ]
        },
        {
            "name": "scorekeep-api",
            "image": "123456789012.dkr.ecr.us-east-2.amazonaws.com/scorekeep-api",
            "cpu": 192,
            "memoryReservation": 512,
            "environment": [
                { "name" : "AWS_REGION", "value" : "us-east-2" },
                { "name" : "NOTIFICATION_TOPIC", "value" : "arn:aws:sns:us-
east-2:123456789012:scorekeep-notifications" }
```

```
            ],
            "portMappings" : [
                {
                    "containerPort": 5000
                }
            ]
        }
    ]
}
```

在 Amazon ECS 控制台中配置命令行选项

命令行选项将覆盖映像配置文件中的任何冲突值。命令行选项通常用于本地测试，但为了方便设置环境变量或控制启动过程，也可以使用命令行选项。

添加命令行选项会更新传递到该容器的 Docker CMD。有关更多信息，请参阅 Docker 运行参考。

设置命令行选项

1. 打开 Amazon ECS 经典控制台，网址为https://console.aws.amazon.com/ecs/。
2. 从导航栏中，选择包含您的任务定义的区域。
3. 在导航窗格中，选择 Task Definitions。
4. 在 Task Definitions 页面上，选择要修订的任务定义左侧的框，然后选择 Create new revision。
5. 在创建任务定义新修订页面上，选择该容器。
6. 在环境部分，将用逗号分隔的命令行选项列表添加到命令字段。
7. 选择更新。
8. 验证信息并选择 Create。

以下示例演示了如何为 RoleARN 选项编写以逗号分隔的命令行选项。RoleARN 选项采用指定的 IAM 角色将分段上传到其他账户。

Example

```
--role-arn, arn:aws:iam::123456789012:role/xray-cross-account
```

要了解有关 X-Ray 中可用命令行选项的更多信息，请参阅配置 AWS X-Ray 守护程序。

AWS X-Ray 与其他人集成 AWS 服务

许多 AWS 服务 提供不同级别的 X-Ray 集成，包括采样和向传入请求添加标头、运行 X-Ray 守护程序以及自动向 X-Ray 发送跟踪数据。与 X-Ray 的集成包括以下内容：

- 主动检测 - 采样和检测传入请求
- 被动检测 - 检测已经由其他服务采样的请求
- 请求跟踪 - 对所有传入请求添加一个跟踪标头，并将其向下游传播
- 工具 - 运行 X-Ray 进程守护程序从 X-Ray SDK 接收分段

> **Note**
>
> X-Ray SDKs 包括用于进一步集成的插件 AWS 服务。例如，您可以将 X-Ray SDK 用于 Java Elastic Beanstalk 插件，以添加有关运行您应用程序的 Elastic Beanstalk 环境的信息（包括环境名称和 ID）。

以下是一些与 X-R AWS 服务 ay 集成的示例：

- AWS Distro for OpenTelemetry (ADOT) — 借助 ADOT，工程师只需对应用程序进行一次检测，即可将相关的指标和跟踪发送到多个监控 AWS 解决方案，包括亚马逊 CloudWatch、亚马逊服务和适用于 Pro AWS X-Ray metheus 的亚马逊托管服务。 OpenSearch
- AWS Lambda— 在所有运行时对传入的请求进行主动和被动检测。 AWS Lambda 向您的追踪地图添加两个节点，一个用于 AWS Lambda 服务，一个用于函数。启用检测后， AWS Lambda 还会在 Java 和 Node.js 运行时上运行 X-Ray 守护程序，以便与 X-Ray SDK 配合使用。
- Amazon API Gateway – 主动和被动检测。API 网关使用采样规则来确定要记录的请求，并向服务地图添加网关阶段的节点。
- AWS Elastic Beanstalk - 工具。在以下平台上，Elastic Beanstalk 包括 X-Ray 进程守护程序：
  - Java SE - 2.3.0 及更高版本的配置
  - Tomcat - 2.4.0 及更高版本的配置
  - Node.js - 3.2.0 及更高版本的配置
  - Windows Server - 除了 2016 年 12 月 9 日起发布的 Windows Server Core 以外的所有配置。

  您可以使用 Elastic Beanstalk 控制台告知 Elastic Beanstalk 在这些平台上运行进程守护程序，或者在 `aws:elasticbeanstalk:xray` 命名空间中使用 `XRayEnabled` 选项。

- Elastic Load Balancin - 应用程序负载均衡器上的请求跟踪。应用程序负载均衡器会将跟踪 ID 添加到请求标头，然后将它发送到目标组。
- 亚马逊 EventBridge — 被动工具。如果使用 X-Ray SDK 对向 EventBridge其发布事件的服务进行检测，则事件目标将收到跟踪标头并可以继续传播原始跟踪 ID。
- Amazon Simple Notification Service - 主动检测。如果 Amazon SNS 发布者使用 X-Ray SDK 跟踪其客户端，订阅者可以检索跟踪标头并继续使用相同的跟踪 ID 传播发布者的原始跟踪。
- Amazon Simple Queue Service - 主动检测。如果某项服务使用 X-Ray 开发工具包跟踪请求，则 Amazon SQS 可以发送跟踪标头并继续使用一致的跟踪 ID 将原始跟踪从发送者传播到使用器。

从以下主题中进行选择，探索全套集成 AWS 服务。

主题

- AWS 和的 OpenTelemetry 发行版 AWS X-Ray
- Amazon API Gateway 主动追踪支持 AWS X-Ray
- 亚马逊 EC2 和 AWS App Mesh
- AWS 应用程序运行器和 X-Ray
- AWS AppSync 和 AWS X-Ray
- 使用记录 X-Ray API 调用 AWS CloudTrail
- CloudWatch 与 X-Ray 集成
- 使用跟踪 X-Ray 加密配置的更改 AWS Config
- Amazon 弹性计算云和 AWS X-Ray
- AWS Elastic Beanstalk 和 AWS X-Ray
- Elastic Load Balancin AWS X-Ray
- 亚马逊 EventBridge 和 AWS X-Ray
- AWS Lambda 和 AWS X-Ray
- 亚马逊 SNS 和 AWS X-Ray
- AWS Step Functions 和 AWS X-Ray
- 亚马逊 SQS 和 AWS X-Ray
- 亚马逊 S3 和 AWS X-Ray

AWS 和的 OpenTelemetry 发行版 AWS X-Ray

使用 AWS Distro for OpenTelemetry (ADOT) 收集指标和跟踪以及其他监控解决方案，例如亚马逊、CloudWatch亚马逊 OpenSearch 服务和适用于 Prometheus 的亚马逊托管服务。 AWS X-Ray

AWS 的发行版 OpenTelemetry

OpenTelemetry (ADOT) AWS 发行版是基于云原生计算基金会 (CNCF) 项目的 AWS 发行版。OpenTelemetry OpenTelemetry 提供一组开源 APIs、库和代理，用于收集分布式跟踪和指标。该工具包是上游 OpenTelemetry 组件的发行版 SDKs，包括经过测试、优化、保护和支持的 AWS自动检测代理和收集器。

借助 ADOT，工程师只需对应用程序进行一次检测，即可将相关的指标和跟踪发送到多个 AWS 监控解决方案，包括亚马逊 CloudWatch、 AWS X-Ray亚马逊 OpenSearch 服务和适用于 Prometheus 的亚马逊托管服务。

ADOT 与越来越多的解决方案集成， AWS 服务 以简化向 X-Ray 等监控解决方案发送跟踪和指标的过程。与 ADOT 集成的一些服务示例包括：

- AWS Lambda— 适用于 ADOT 的 AWS 托管 Lambda 层通过自动检测 Lambda 函数，将 AWS Lambda 和 X-Ray 的 out-of-the-box配置一起 OpenTelemetry 打包到易于设置的层中，从而提供 plug-and-play用户体验。用户无需更改代码即可启用和禁 OpenTelemetry 用其 Lambda 函数。有关更多信息，请参阅适用于 Lamb AWS da 的 OpenTelemetry 发行版
- 亚马逊弹性容器服务 (ECS) — 使用 AWS Distro for Collector Collector 从亚马逊 ECS 应用程序 OpenTelemetry 收集指标和跟踪，然后发送到 X-Ray 和其他监控解决方案。有关更多信息，请参阅《Amazon ECS 开发人员指南》中的收集应用程序跟踪数据。
- AWS App Runner — App Runner 支持使用 AWS Distro for OpenTelemetry (ADOT) 向 X-Ray 发送跟踪。使用 ADOT SDKs 收集容器化应用程序的跟踪数据，并使用 X-Ray 分析您的仪器化应用程序并获得对这些应用程序的见解。有关更多信息，请参阅 AWS App Runner 和 X-Ray。

有关发行版的更多信息 OpenTelemetry，包括与其他 AWS 发行版的集成 AWS 服务，请参阅文档AWS 发行版。 OpenTelemetry

有关使用 Distro for 和 OpenTelemetry X-Ray 对应用程序进行检测的更多信息，请参阅使用 AWS Distro 对应用程序进行检测。 AWS OpenTelemetry

Amazon API Gateway 主动追踪支持 AWS X-Ray

当用户请求通过您的 Amazon API Gateway 传送到底层服务时，您可以使用 X-Ray APIs 来跟踪和分析这些请求。API Gateway 支持对所有 API Gateway 端点类型进行 X-Ray 跟踪：区域、边缘优化和私有。在所有可用 X-Ray AWS 区域 的地方，你都可以在亚马逊 API Gateway 上使用 X-Ray。有关更多信息，请参阅《API Gateway 开发人员指南》中的使用 AWS X-Ray跟踪 API Gateway API 执行情况。

> **Note**
>
> X-Ray 仅支持 APIs 通过 API Gateway 跟踪 REST。

Amazon API Gateway 为提供主动追踪支持 AWS X-Ray。在 API 阶段启用活动跟踪以对传入请求进行采样，并将跟踪发送到 X-Ray。

如何在 API 阶段启用活动跟踪

1. 打开 API Gateway 控制台，网址为https://console.aws.amazon.com/apigateway/。
2. 选择一个 API。
3. 选择一个阶段。
4. 在日志/跟踪选项卡上，选择启用 X-Ray 跟踪，然后选择保存更改。
5. 在左侧导航面板中，选择资源。
6. 如需重新部署具有新设置的 API，请选择操作下拉列表，然后选择部署 API。

API Gateway 使用您在 X-Ray 控制台中定义的采样规则来确定要记录的请求。您可以创建仅适用于或仅适用于 APIs包含特定标头的请求的规则。API Gateway 在分段的属性中记录标头，以及有关阶段和请求的详细信息。有关更多信息，请参阅 配置采样规则。

> **Note**
>
> APIs 使用 API Gateway HTTP 集成跟踪 REST 时，每个分段的服务名称都设置为从 API Gateway 到您的 HTTP 集成端点的请求网址路径，从而在 X-Ray 跟踪地图上为每个唯一网址路径生成一个服务节点。大量 URL 路径可能会导致跟踪地图超过 10,000 节点的上限，从而出现错误。

如需最大限度减少 API Gateway 创建的服务节点数量，请考虑在 URL 查询字符串或通过 POST 的请求正文里传递参数。这两种方法都能确保参数不是 URL 路径的一部分，这可能会减少不同的 URL 路径和服务节点。

对于所有传入请求，API Gateway 将跟踪标头添加到还没有跟踪标头的传入 HTTP 请求。

```
X-Amzn-Trace-Id: Root=1-5759e988-bd862e3fe1be46a994272793
```

X-Ray 跟踪 ID 格式

X-Ray `trace_id` 由以连字符分隔的三组数字组成。例如，`1-58406520-a006649127e371903a2de979`。这包括：

- 版本号，即 1。
- 原始请求的时间，采用 Unix 纪元时间，为 8 个十六进制数字。

  例如，2016 年 12 月 1 日上午 10:00（太平洋标准时间）的纪元时间为 1480615200 秒，或者是十六进制数字 58406520。
- 跟踪的 96 位全局唯一标识符，使用 24 个十六进制数字。

如果禁用了活动跟踪，只要请求来自采样的请求并已开始跟踪，则该阶段仍会记录分段。例如，已检测的 Web 应用程序可通过 HTTP 客户端调用 API Gateway API。当您使用 X-Ray SDK 检测 HTTP 客户端时，将向包含采样决策的传出请求添加跟踪标头。API Gateway 读取跟踪标头并为采样请求创建分段。

如果您使用 API Gateway 为您的 API 生成 Java SDK，则可以通过在客户端生成器中添加请求处理程序来检测 SDK 客户端，就像手动检测 AWS SDK 客户端一样。有关说明，请参阅使用适用于 Java 的 X-Ray SD AWS K 追踪 SDK 调用：

亚马逊 EC2 和 AWS App Mesh

AWS X-Ray 与集成AWS App Mesh以管理微服务的 Envoy 代理。App Mesh 提供了 Envoy 版本，您可以将其配置为向在相同任务或 pod 的容器中运行的 X-Ray 进程守护程序发送跟踪数据。X-Ray 支持使用以下与 App Mesh 兼容的服务进行跟踪：

- Amazon Elastic Container Service (Amazon ECS)

- Amazon Elastic Kubernetes Service (Amazon EKS)
- 亚马逊弹性计算云（亚马逊 EC2）

使用以下说明以了解如何通过 App Mesh 启用 X-Ray 跟踪。

要配置 Envoy 代理以将数据发送到 X-Ray，请在其容器定义中设置 `ENABLE_ENVOY_XRAY_TRACING` 环境变量。

> **Note**
>
> Envoy 的 App Mesh 版本目前不根据配置的采样规则发送跟踪。而是使用 5% 的固定采样率（针对 Envoy 版本 1.16.3 或更新版本），或 50% 的采样率（针对 Envoy 1.16.3 之前的版本）。

Example Amazon ECS 的 Envoy 容器定义

```
{
      "name": "envoy",
      "image": "public.ecr.aws/appmesh/aws-appmesh-envoy:envoy-version",
      "essential": true,
      "environment": [
        {
          "name": "APPMESH_VIRTUAL_NODE_NAME",
          "value": "mesh/myMesh/virtualNode/myNode"
        },
        {
          "name": "ENABLE_ENVOY_XRAY_TRACING",
          "value": "1"
        }
      ],
      "healthCheck": {
        "command": [
          "CMD-SHELL",
          "curl -s http://localhost:9901/server_info | cut -d' ' -f3 | grep -q live"
        ],
        "startPeriod": 10,
        "interval": 5,
        "timeout": 2,
        "retries": 3
      }
```

> **Note**
>
> 请参阅《AWS App Mesh 用户指南》中的 Envoy 图像，了解有关可用 Envoy 区域地址的更多信息。

有关在容器中运行 X-Ray 进程守护程序的详细信息，请参阅 在 Amazon ECS 上运行 X-Ray 进程守护程序。对于包含服务网格、微服务、Envoy 代理和 X-Ray 守护程序的示例应用程序，请在 App Mesh colorapp 示例存储库中部署示例 GitHub。

了解更多

- AWS App Mesh入门
- 入门 AWS App Mesh 和 Amazon ECS

AWS 应用程序运行器和 X-Ray

AWS App Runner 提供了 AWS 服务 一种快速、简单且经济实惠的方式，可以将源代码或容器映像直接部署到中可扩展且安全的 Web 应用程序 AWS 云。您无需学习新技术、决定使用哪种计算服务，也不需要知道如何预置和配置 AWS 资源。有关更多信息，请参阅什么是 AWS App Runner。

AWS App Runner 通过与 AWS Distro for OpenTelemetry (ADOT) 集成，向 X-Ray 发送跟踪。使用 ADOT SDKs 收集容器化应用程序的跟踪数据，并使用 X-Ray 分析您的仪器化应用程序并获得对这些应用程序的见解。有关更多信息，请参阅使用 X-Ray 跟踪 App Runner 应用程序。

AWS AppSync 和 AWS X-Ray

您可以启用和跟踪的请求 AWS AppSync。有关更多信息，请参阅使用 AWS X-Ray 进行追踪以了解相关说明。

为某 AWS AppSync 个 API 启用 X-Ray 跟踪后，系统会在您的账户中自动创建具有相应权限的 AWS 身份和访问管理服务相关角色。这允许 AWS AppSync 以安全的方式向 X-Ray 发送跟踪。

使用记录 X-Ray API 调用 AWS CloudTrail

AWS X-Ray 与AWS CloudTrail一项服务集成，该服务提供用户、角色或. 所执行操作的记录 AWS 服务。 CloudTrail 将 X-Ray 的所有 API 调用捕获为事件。捕获的调用包括来自 X-Ray 控制台的调用和对 X-Ray API 操作的代码调用。使用收集的信息 CloudTrail，您可以确定向 X-Ray 发出的请求、发出请求的 IP 地址、发出请求的时间以及其他详细信息。

每个事件或日志条目都包含有关生成请求的人员信息。身份信息有助于您确定以下内容：

- 请求是使用根用户凭证还是用户凭证发出的。
- 请求是否代表 IAM Identity Center 用户发出。

- 请求是使用角色还是联合用户的临时安全凭证发出的。
- 请求是否由其他 AWS 服务发出。

CloudTrail 在您创建账户 AWS 账户 时在您的账户中处于活动状态，并且您自动可以访问 CloudTrail 活动历史记录。 CloudTrail 事件历史记录提供了过去 90 天中记录的管理事件的可查看、可搜索、可下载且不可变的记录。 AWS 区域有关更多信息，请参阅《AWS CloudTrail 用户指南》中的 “使用 CloudTrail 事件历史记录”。查看活动历史记录不 CloudTrail收取任何费用。

要持续记录 AWS 账户 过去 90 天内的事件，请创建跟踪或 CloudTrailLake 事件数据存储。

CloudTrail 步道

跟踪允许 CloudTrail 将日志文件传输到 Amazon S3 存储桶。使用创建的所有跟踪 AWS Management Console 都是多区域的。您可以通过使用 AWS CLI创建单区域或多区域跟踪。建议创建多区域跟踪，因为您可以捕获账户 AWS 区域 中的所有活动。如果您创建单区域跟踪，则只能查看跟踪的 AWS 区域中记录的事件。有关跟踪的更多信息，请参阅《AWS CloudTrail 用户指南》中的为您的 AWS 账户创建跟踪和为组织创建跟踪。

通过创建跟踪，您可以免费将正在进行的管理事件的一份副本传送到您的 Amazon S3 存储桶，但会收取 Amazon S3 存储费用。 CloudTrail 有关 CloudTrail 定价的更多信息，请参阅AWS CloudTrail 定价。有关 Amazon S3 定价的信息，请参阅 Amazon S3 定价。

CloudTrail 湖泊事件数据存储

CloudTrail L@@ ak e 允许您对事件运行基于 SQL 的查询。 CloudTrail Lake 将基于行的 JSON 格式的现有事件转换为 Apache ORC 格式。ORC 是一种针对快速检索数据进行优化的列式存储格式。事件将被聚合到事件数据存储中，它是基于您通过应用高级事件选择器选择的条件的不可变的事件集合。应用于事件数据存储的选择器用于控制哪些事件持续存在并可供您查询。有关 CloudTrail Lake 的更多信息，请参阅AWS CloudTrail 用户指南中的使用 AWS CloudTrail Lake。

CloudTrail 湖泊事件数据存储和查询会产生费用。创建事件数据存储时，您可以选择要用于事件数据存储的定价选项。定价选项决定了摄取和存储事件的成本，以及事件数据存储的默认和最长保留期。有关 CloudTrail 定价的更多信息，请参阅AWS CloudTrail 定价。

主题

- 中的 X-Ray 管理事件 CloudTrail
- 中的 X-ray 数据事件 CloudTrail
- X-Ray 事件示例

中的 X-Ray 管理事件 CloudTrail

AWS X-Ray 与集成 AWS CloudTrail 以记录用户、角色或用户 AWS 服务 在 X-Ray 中执行的 API 操作。您可以使用 CloudTrail 实时监控 X-Ray API 请求并将日志存储在 Amazon S3、亚马逊 CloudWatch 日志和亚马逊 CloudWatch 事件中。X-Ray 支持将以下操作作为事件 CloudTrail 记录在日志文件中：

支持的 API 操作

- PutEncryptionConfig
- GetEncryptionConfig
- CreateGroup
- UpdateGroup
- DeleteGroup
- GetGroup
- GetGroups
- GetInsight
- GetInsightEvents
- GetInsightImpactGraph
- GetInsightSummaries
- GetSamplingStatisticSummaries

中的 X-ray 数据事件 CloudTrail

数据事件提供有关在资源上或在资源中执行的资源操作的信息（例如，PutTraceSegments，它将分段文档上传到 X-Ray）。

这些也称为数据层面操作。数据事件通常是高容量活动。默认情况下， CloudTrail 不记录数据事件。CloudTrail 事件历史记录不记录数据事件。

记录数据事件将收取额外费用。有关 CloudTrail 定价的更多信息，请参阅AWS CloudTrail 定价。

您可以使用 CloudTrail 控制台、 AWS CLI或 CloudTrail API 操作记录 X-Ray 资源类型的数据事件。有关如何记录数据事件的更多信息，请参阅《AWS CloudTrail 用户指南》中的使用 AWS Management Console记录数据事件和使用 AWS Command Line Interface记录数据事件。

下表列出了您可以为其记录数据事件的 X-Ray 资源类型。数据事件类型（控制台）列显示要从控制 CloudTrail 台上的数据事件类型列表中选择的值。res ources.type 值列显示该`resources.type`值，您将在使用或配置高级事件选择器时指定该值。 AWS CLI CloudTrail APIs“ APIs 记录到的数据 CloudTrail” 列显示了 CloudTrail 针对该资源类型记录的 API 调用。

数据事件类型（控制台）	resources.type 值	数据 APIs 已记录到 CloudTrail
X-Ray 跟踪	`AWS::XRay::Trace`	• PutTraceSegments • GetTraceSummaries • GetTraceGraph • GetServiceGraph • BatchGetTraces • GetTimeSeriesServiceStatistics • PutTelemetryRecords • GetSamplingTargets

您可以将高级事件选择器配置为在 `eventName` 和 `readOnly` 字段上进行筛选，从而仅记录那些对您很重要的事件。但是，您无法通过添加`resources.ARN`字段选择器来选择事件，因为 X-Ray 轨迹没有 ARNs。有关这些字段的更多信息，请参阅 AdvancedFieldSelector《AWS CloudTrail API 参考》中的 。以下是如何运行`put-event-selectors` AWS CLI 命令以记录 CloudTrail 跟踪上的数据事件的示例。您必须在其中运行命令或指定创建跟踪的区域；否则，该操作将返回 `InvalidHomeRegionException` 异常。

```
aws cloudtrail put-event-selectors --trail-name myTrail --advanced-event-selectors \
'{
   "AdvancedEventSelectors": [
      {
         "FieldSelectors": [
            { "Field": "eventCategory", "Equals": ["Data"] },
            { "Field": "resources.type", "Equals": ["AWS::XRay::Trace"] },
            { "Field": "eventName", "Equals":
 ["PutTraceSegments","GetSamplingTargets"] }
         ],
         "Name": "Log X-Ray PutTraceSegments and GetSamplingTargets data events"
      }
   ]
```

```
}'
```

X-Ray 事件示例

管理事件示例 **GetEncryptionConfig**

以下是 X-Ray 的示例 GetEncryptionConfig 登录条目 CloudTrail。

Example

```
{
    "eventVersion"=>"1.05",
    "userIdentity"=>{
        "type"=>"AssumedRole",
        "principalId"=>"AROAJVHBZWD3DN6CI2MHM:MyName",
        "arn"=>"arn:aws:sts::123456789012:assumed-role/MyRole/MyName",
        "accountId"=>"123456789012",
        "accessKeyId"=>"AKIAIOSFODNN7EXAMPLE",
        "sessionContext"=>{
            "attributes"=>{
                "mfaAuthenticated"=>"false",
                "creationDate"=>"2023-7-01T00:24:36Z"
            },
            "sessionIssuer"=>{
                "type"=>"Role",
                "principalId"=>"AROAJVHBZWD3DN6CI2MHM",
                "arn"=>"arn:aws:iam::123456789012:role/MyRole",
                "accountId"=>"123456789012",
                "userName"=>"MyRole"
            }
        }
    },
    "eventTime"=>"2023-7-01T00:24:36Z",
    "eventSource"=>"xray.amazonaws.com",
    "eventName"=>"GetEncryptionConfig",
    "awsRegion"=>"us-east-2",
    "sourceIPAddress"=>"33.255.33.255",
    "userAgent"=>"aws-sdk-ruby2/2.11.19 ruby/2.3.1 x86_64-linux",
    "requestParameters"=>nil,
    "responseElements"=>nil,
    "requestID"=>"3fda699a-32e7-4c20-37af-edc2be5acbdb",
    "eventID"=>"039c3d45-6baa-11e3-2f3e-e5a036343c9f",
    "eventType"=>"AwsApiCall",
```

```
    "recipientAccountId"=>"123456789012"
}
```

数据事件示例 PutTraceSegments

以下是 X-Ray 的示例 PutTraceSegments 中的数据事件日志条目 CloudTrail。

Example

```
{
  "eventVersion": "1.09",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AROAWYXPW54Y4NEXAMPLE:i-0dzz2ac111c83zz0z",
    "arn": "arn:aws:sts::012345678910:assumed-role/my-service-role/
i-0dzz2ac111c83zz0z",
    "accountId": "012345678910",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AROAWYXPW54Y4NEXAMPLE",
        "arn": "arn:aws:iam::012345678910:role/service-role/my-service-role",
        "accountId": "012345678910",
        "userName": "my-service-role"
      },
      "attributes": {
        "creationDate": "2024-01-22T17:34:11Z",
        "mfaAuthenticated": "false"
      },
      "ec2RoleDelivery": "2.0"
    }
  },
  "eventTime": "2024-01-22T18:22:05Z",
  "eventSource": "xray.amazonaws.com",
  "eventName": "PutTraceSegments",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "198.51.100.0",
  "userAgent": "aws-sdk-ruby3/3.190.0 md/internal ua/2.0 api/xray#1.0.0 os/linux md/
x86_64 lang/ruby#2.7.8 md/2.7.8 cfg/retry-mode#legacy",
  "requestParameters": {
    "traceSegmentDocuments": [
      "trace_id:1-00zzz24z-EXAMPLE4f4e41754c77d0000",
```

```
      "trace_id:1-00zzz24z-EXAMPLE4f4e41754c77d0000",
      "trace_id:1-00zzz24z-EXAMPLE4f4e41754c77d0001",
      "trace_id:1-00zzz24z-EXAMPLE4f4e41754c77d0002"
    ]
  },
  "responseElements": {
    "unprocessedTraceSegments": []
  },
  "requestID": "5zzzzz64-acbd-46ff-z544-451a3ebcb2f8",
  "eventID": "4zz51z7z-77f9-44zz-9bd7-6c8327740f2e",
  "readOnly": false,
  "resources": [
    {
      "type": "AWS::XRay::Trace"
    }
  ],
  "eventType": "AwsApiCall",
  "managementEvent": false,
  "recipientAccountId": "012345678910",
  "eventCategory": "Data",
  "tlsDetails": {
    "tlsVersion": "TLSv1.2",
    "cipherSuite": "ZZZZZ-RSA-AAA128-GCM-SHA256",
    "clientProvidedHostHeader": "example.us-west-2.xray.cloudwatch.aws.dev"
  }
}
```

CloudWatch 与 X-Ray 集成

AWS X-Ray 与CloudWatch 应用程序信号、 CloudWatch RUM 和 Syn CloudWatch thetics 集成，可以更轻松地监控应用程序的运行状况。为 Application Signals 启用应用程序，以监控服务、客户端页面、Synthetics Canary 和服务依赖项的运行状况并对其进行问题排查。

通过关联 CloudWatch 指标、日志和 X-Ray 跟踪，X-Ray 跟踪地图可提供您的服务 end-to-end视图，帮助您快速查明性能瓶颈并识别受影响的用户。

借 CloudWatch 助 RUM，您可以执行真实的用户监控，以近乎实时的方式收集和查看来自实际用户会话的 Web 应用程序性能的客户端数据。借助 AWS X-Ray 和 CloudWatch RUM，您可以分析和调试从应用程序的最终用户到下游 AWS 托管服务的请求路径。可帮助您识别影响最终用户的延迟趋势和错误。

主题

CloudWatch 朗姆酒和 AWS X-Ray

借助 Amazon CloudWatch RUM，您可以执行真实用户监控，以近乎实时的方式收集和查看来自实际用户会话的 Web 应用程序性能的客户端数据。借助 AWS X-Ray 和 CloudWatch RUM，您可以分析和调试从应用程序的最终用户到下游 AWS 托管服务的请求路径。可帮助您识别影响最终用户的延迟趋势和错误。

开启用户会话的 X-Ray 跟踪后， CloudWatch RUM 会向允许的 HTTP 请求添加一个 X-Ray 跟踪标头，并记录允许的 HTTP 请求的 X-Ray 分段。然后，您可以在 X-Ray 和 CloudWatch 控制台中查看来自这些用户会话的轨迹和区段，包括 X-Ray 跟踪地图。

> **Note**
>
> CloudWatch RUM 未与 X-Ray 采样规则集成。相反，在将应用程序设置为使用 CloudWatch RUM 时，请选择采样百分比。从 R CloudWatch UM 发送的跟踪可能会产生额外费用。有关更多信息，请参阅 AWS X-Ray 定价。

默认情况下，从 CloudWatch RUM 发送的客户端跟踪未连接到服务器端跟踪。要将客户端跟踪与服务器端跟踪连接起来，请将 CloudWatch RUM Web 客户端配置为向这些 HTTP 请求添加 X-Ray 跟踪标头。

> **Warning**
>
> 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域资源共享 (CORS) 失败。为避免这种情况，请将 `X-Amzn-Trace-Id` HTTP 标头添加到下游服务 CORS 配置的允许标头列表中。如果您使用 API Gateway 作为下游，请参阅为 REST API 资源启用 CORS。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。

有关中真实用户监控的更多信息 CloudWatch，请参阅使用 CloudWatch RUM。要将应用程序设置为使用 CloudWatch RUM，包括使用 X-Ray 跟踪用户会话，请参阅将应用程序设置为使用 CloudWatch RUM。

使用 X-Ra CloudWatch y 调试合成金丝雀

CloudWatch Synthetics 是一项完全托管的服务，可让您监控端点并 APIs 使用每天 24 小时、每分钟运行一次的脚本化金丝雀。

您可以自定义 Canary 脚本以检查以下内容中的更改：

- 可用性
- 延迟
- 事务
- 中断或失效的链接
- Step-by-step 任务完成
- 页面加载错误
- UI 资产的加载延迟
- 复杂的向导流
- 应用程序中的结算流程

Canary 遵循与客户相同的路线执行相同的操作和行为，并不断验证客户体验。

要了解有关设置 Synthetics 测试的详细信息，请参阅使用 Synthetics 创建和管理 Canary。

以下示例显示 Synthetics Canary 引起的调试问题的常见使用案例。每个示例都演示了使用跟踪地图或 X-Ray Analytics 控制台进行调试的关键策略。

有关如何解读跟踪地图并与之进行互动的更多信息，请参阅查看服务地图。

有关如何阅读 X-Ray Analytics 控制台并与之交互的更多信息，请参阅与 An AWS X-Ray alytics 控制台交互。

主题

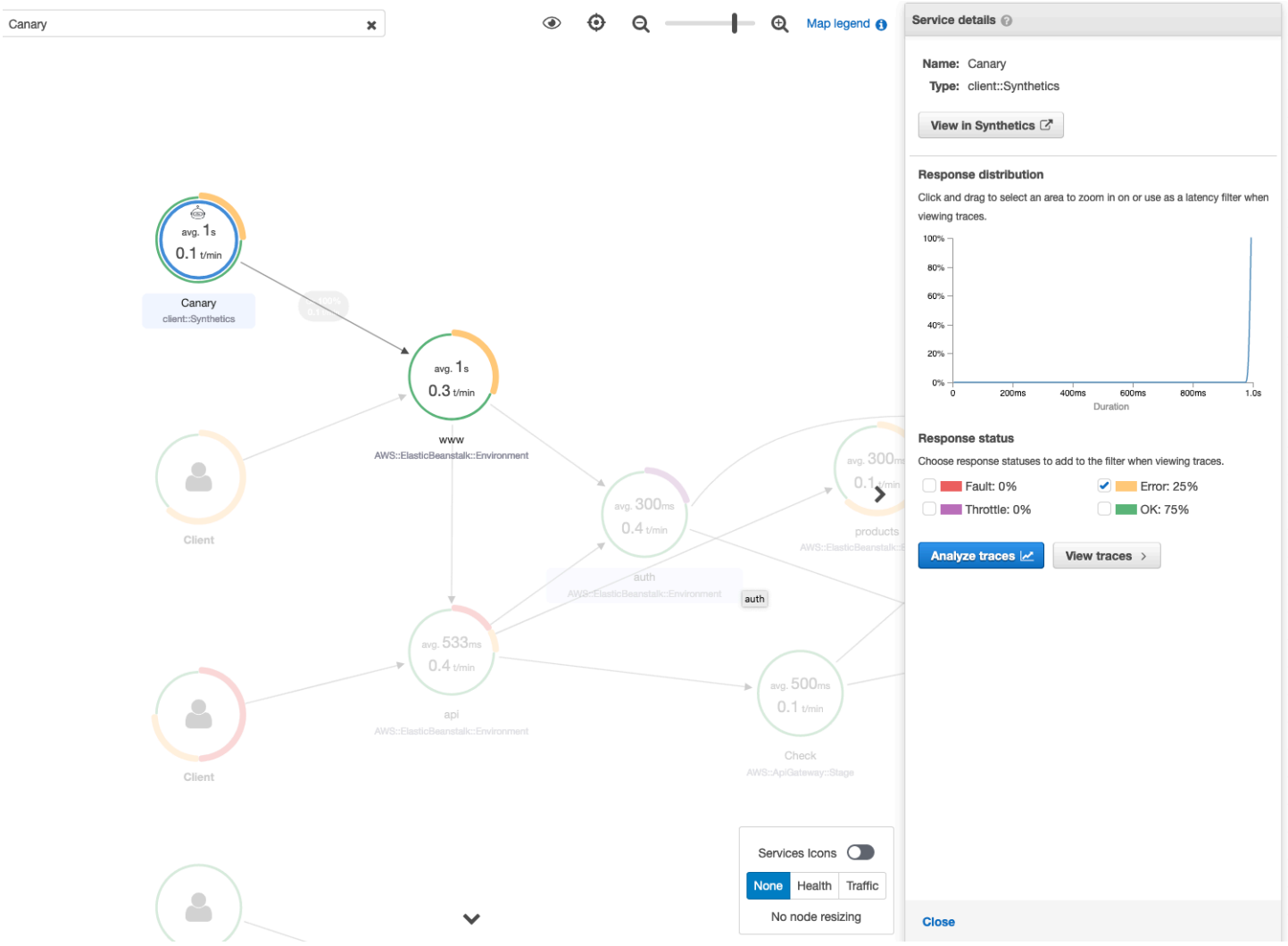
- 在跟踪地图中查看带有增强错误报告的 Canary
- 对各个跟踪使用跟踪详情地图以详细查看每个请求
- 确定上游和下游服务持续出现故障的根本原因

- 确定性能瓶颈和趋势
- 比较更改前后的延迟和错误或故障率
- 确定所有人所需的金丝雀覆盖范围 APIs 以及 URLs
- 使用组专注于 Synthetics 测试

在跟踪地图中查看带有增强错误报告的 Canary

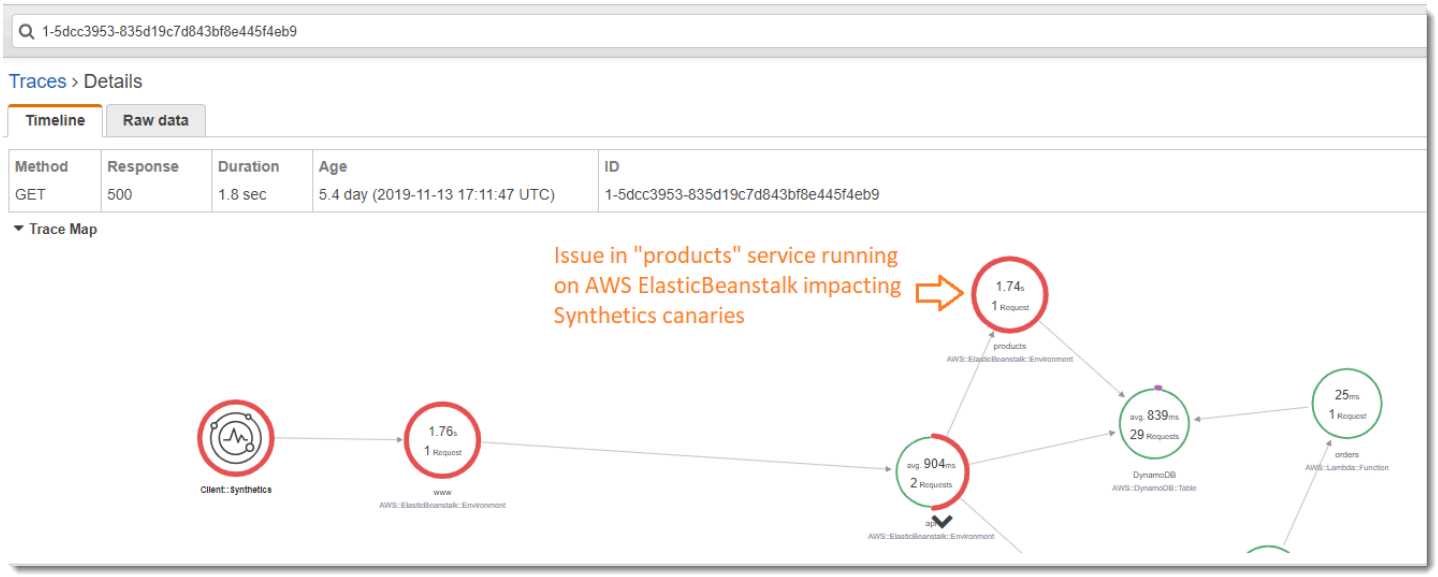
要查看 X-Ray 跟踪地图中哪些 Canary 的错误、故障、限制速率或缓慢响应时间有所增加，您可以使用 `Client::Synthetic` 筛选器突出显示 Synthetics Canary 客户端节点。单击节点将显示整个请求的响应时间分布。单击两个节点之间的边缘会显示有关通过该连接的请求的详细信息。您还可以在跟踪地图中查看相关下游服务的“远程”推断节点。

单击 Synthetics 节点时，侧面板会有一个在 Synthetics 中查看按钮会将您重定向到可在其中查看 Canary 详细信息的 Synthetics 控制台中。



对各个跟踪使用跟踪详情地图以详细查看每个请求

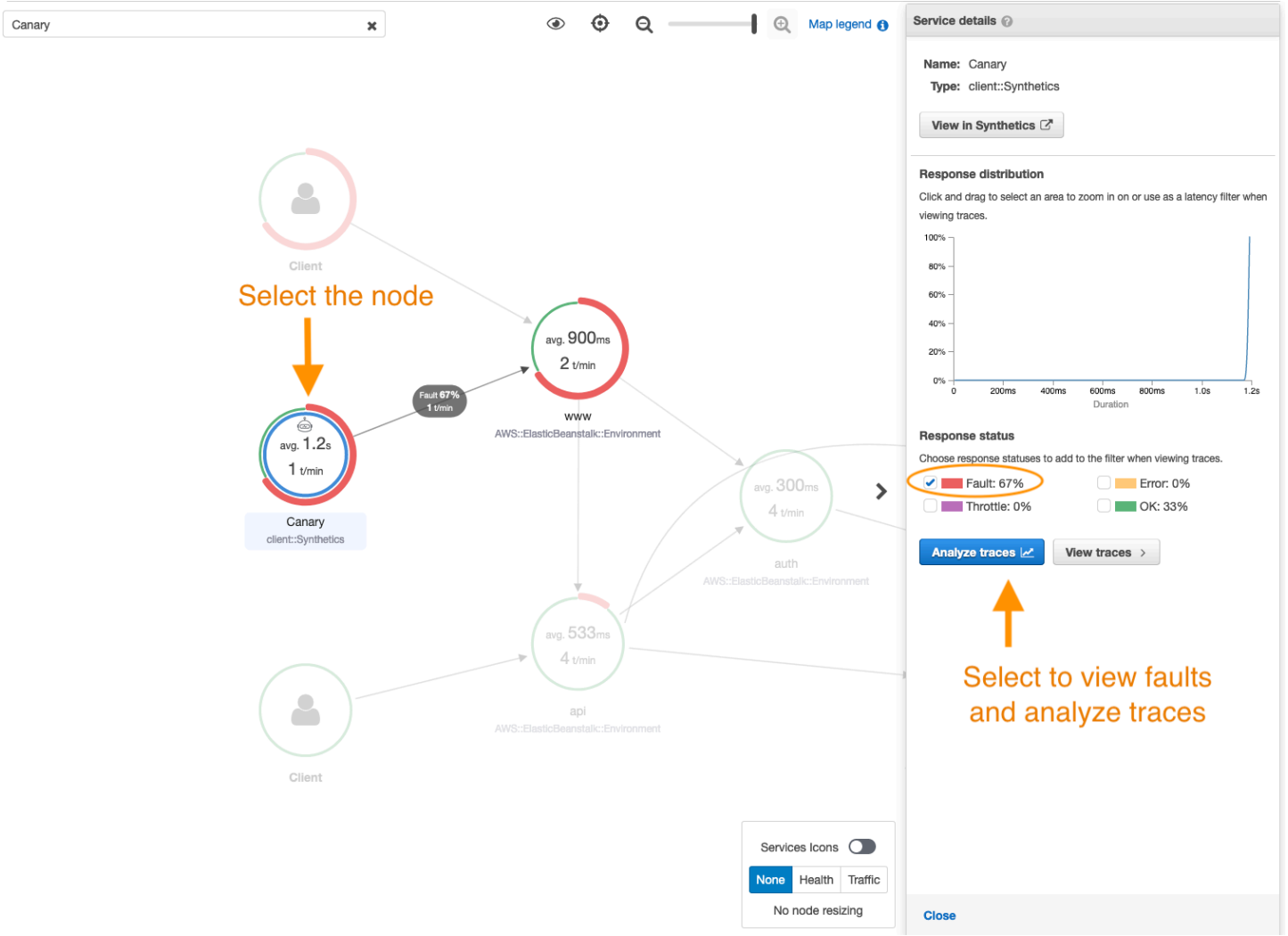
要确定哪些服务会导致最长延迟或导致错误，请通过在跟踪地图中选择跟踪来调用跟踪详情地图。单个跟踪详细信息地图显示单个请求的 end-to-end路径。使用此方法可了解调用的服务，并直观显示上游和下游服务。

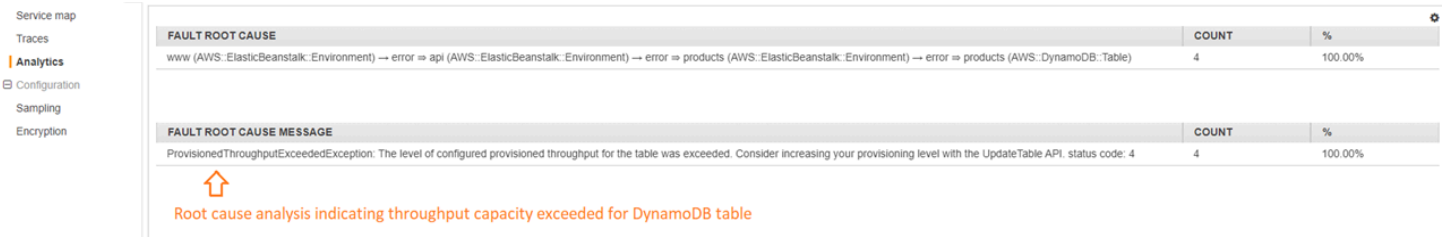


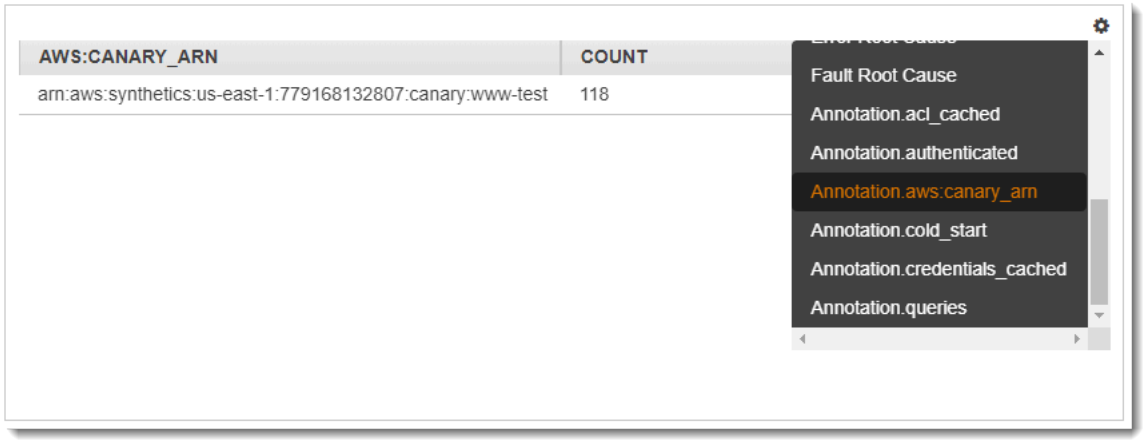
确定上游和下游服务持续出现故障的根本原因

在 Synthetics 金丝雀中收到故障 CloudWatch 警报后，请在 X-Ray 中使用跟踪数据的统计建模在 X-Ray Analytics 控制台中确定问题的可能根本原因。在 Analytics 控制台中，响应时间根本原因表显示了记录的实体路径。X-Ray 确定跟踪中的哪个路径是响应时间的最可能原因。格式指示所遇到的实体的层次结构，结尾是响应时间根本原因。

以下示例显示，由于 Amazon DynamoDB 表中的吞吐量容量异常，对在 API 网关上运行的 API“XXX”进行的 Synthetics 测试失败。

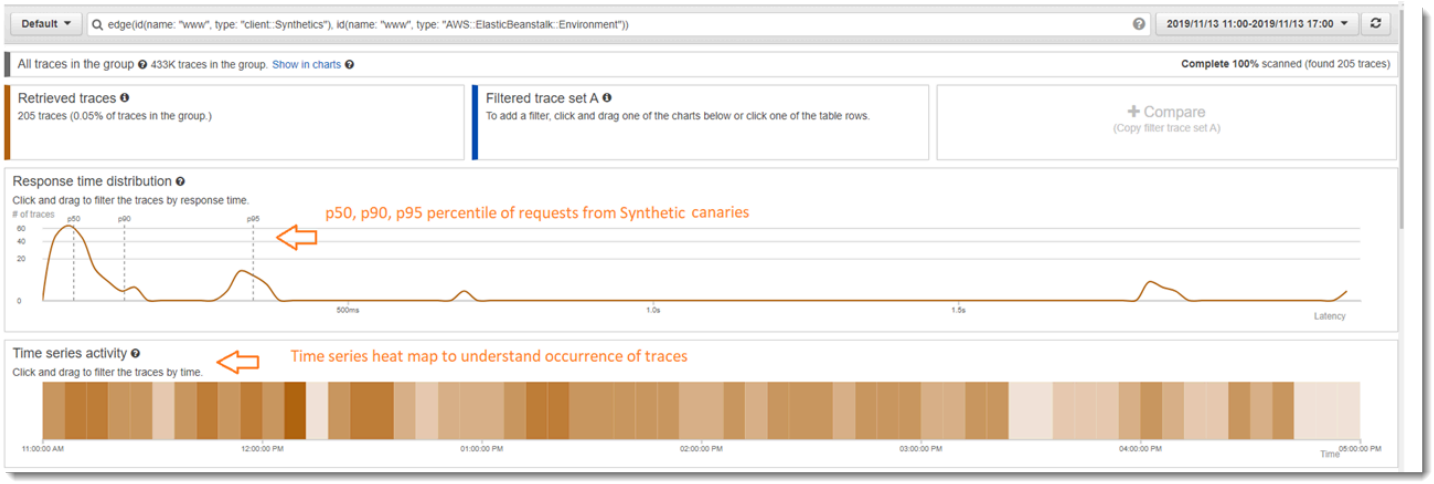

Canary
Map legend
Service details
Name: Canary
Type: client::Synthetics
View in Synthetics
Response distribution
Click and drag to select an area to zoom in on or use as a latency filter when viewing traces.
100%
80%
60%
40%
20%
0%
0
200ms
400ms
600ms
800ms
1.0s
1.2s
Duration
Response status
Choose response statuses to add to the filter when viewing traces.
Fault: 67%
Error: 0%
Throttle: 0%
OK: 33%
Analyze traces
View traces
Select to view faults and analyze traces
Client
Select the node
avg. 900ms
2 t/min
Fault 67%
1 t/min
www
AWS::ElasticBeanstalk::Environment
avg. 1.2s
1 t/min
Canary
client::Synthetics
avg. 300ms
4 t/min
auth
AWS::ElasticBeanstalk::Environment
avg. 533ms
4 t/min
api
AWS::ElasticBeanstalk::Environment
Client
Services Icons
None
Health
Traffic
No node resizing
Close


Service map
Traces
Analytics
Configuration
Sampling
Encryption
FAULT ROOT CAUSE
COUNT
%
www (AWS::ElasticBeanstalk::Environment) ⟶ error ⇒ api (AWS::ElasticBeanstalk::Environment) ⟶ error ⇒ products (AWS::ElasticBeanstalk::Environment) ⟶ error ⇒ products (AWS::DynamoDB::Table)
4
100.00%
FAULT ROOT CAUSE MESSAGE
COUNT
%
ProvisionedThroughputExceededException: The level of configured provisioned throughput for the table was exceeded. Consider increasing your provisioning level with the UpdateTable API. status code: 4
4
100.00%
Root cause analysis indicating throughput capacity exceeded for DynamoDB table



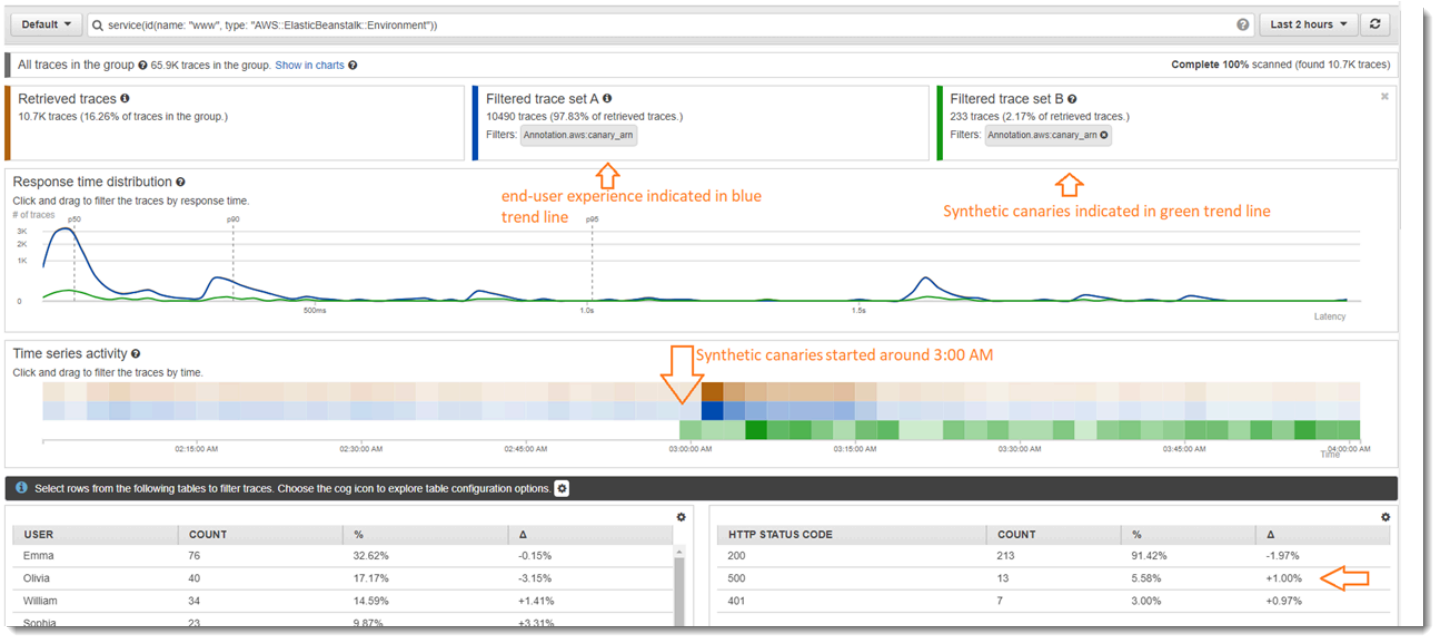
确定性能瓶颈和趋势

您可以使用来自 Synthetics Canary 的持续流量在一段时间内填充跟踪详情地图，从而查看端点性能随时间的趋势。



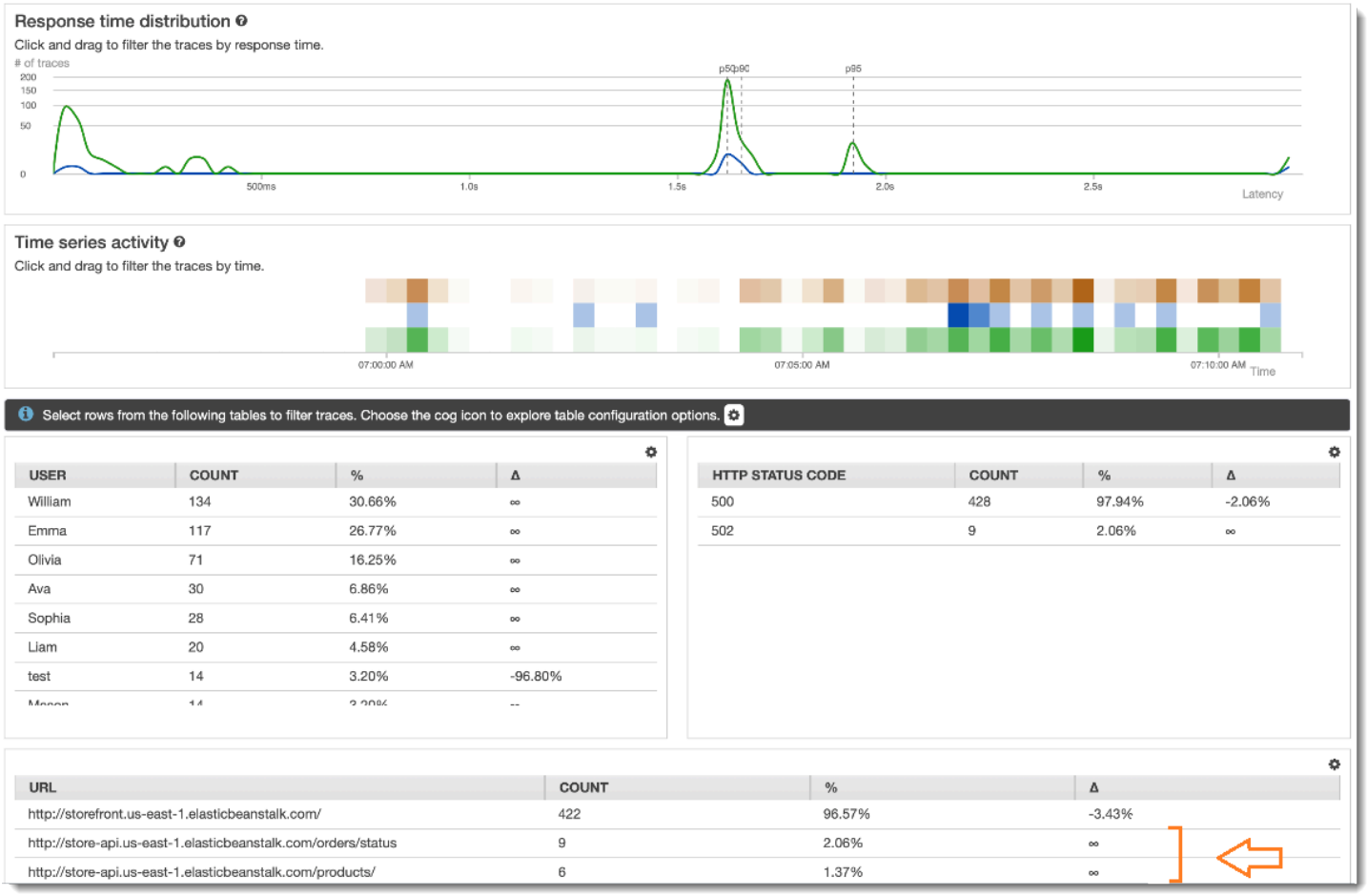
比较更改前后的延迟和错误或故障率

精确确定发生更改的时间，以便将该更改与您的 Canary 捕获的问题增加相关联。使用 X-Ray Analytics 控制台将之前和之后的时间范围定义为不同的跟踪集，从而在响应时间分布中创建视觉差异。



确定所有人所需的金丝雀覆盖范围 APIs 以及 URLs

使用 X-Ray Analytics 与用户比较 Canary 的体验。以下 UI 显示的蓝色趋势线代表 Canary，绿线代表用户。您还可以确定三者中有两个 URLs 没有金丝雀测试。

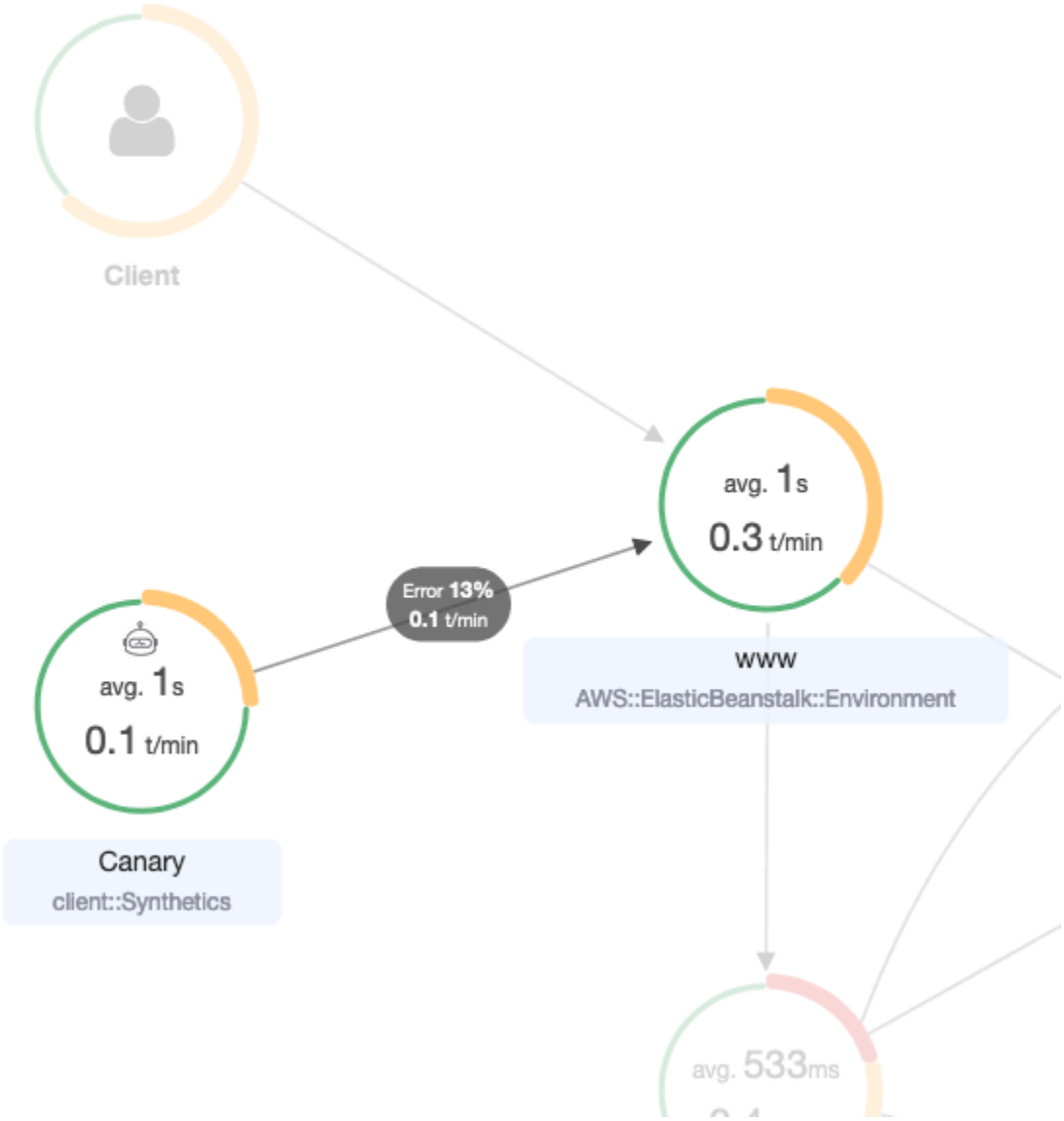


使用组专注于 Synthetics 测试

您可以使用筛选条件表达式创建 X-Ray 组以专注于某组工作流程，例如，对正在 AWS Elastic Beanstalk上运行的“www”进行 Synthetics 测试。使用复杂关键字 `service()` 和 `edge()` 来通过服务和边缘筛选。

Example 组筛选表达式

```
"edge(id(name: "www", type: "client::Synthetics"), id(name: "www", type:
 "AWS::ElasticBeanstalk::Environment"))"
```



使用跟踪 X-Ray 加密配置的更改 AWS Config

AWS X-Ray 与集成 AWS Config 以记录对 X-Ray 加密资源所做的配置更改。您可以使用 AWS Config 清点 X-Ray 加密资源、审计 X-Ray 配置历史记录以及根据资源更改发送通知。

AWS Config 支持将以下 X-Ray 加密资源更改记录为事件：

- 配置更改 - 更改或添加一个加密密钥，或恢复为默认 X-Ray 加密设置。

按照以下说明学习如何在 X-Ray 和之间创建基本连接 AWS Config。

创建 Lambda 函数触发器

在生成自定义规则之前，您必须拥有自定义 AWS Lambda 函数的 ARN。 AWS Config 按照以下说明，通过 Node.js 创建一个基本函数，该函数基于 `XrayEncryptionConfig` 资源的状态将合规或不合规值返回给 AWS Config 。

使用更改触发器创建 Lambda 函数 AWS::Xray EncryptionConfig

1. 打开 Lambda 控制台。选择 Create function（创建函数）。
2. 选择蓝图，然后筛选蓝图的蓝图库。config-rule-change-triggered单击蓝图名称中的链接，或选择配置以继续。
3. 定义以下字段来配置蓝图：
   - 对于名称，键入名称。
   - 对于角色，请选择从模板创建新角色。
   - 对于 Role name，请输入名称。
   - 对于策略模板，选择 AWS Config 规则权限。
4. 选择创建函数以在 AWS Lambda 控制台中创建和显示您的函数。
5. 编辑您的函数代码，将 `AWS::EC2::Instance` 替换为 `AWS::XrayEncryptionConfig`。您还可以更新描述字段来反映此更改。

   默认代码

```
    if (configurationItem.resourceType !== 'AWS::EC2::Instance') {
        return 'NOT_APPLICABLE';
    } else if (ruleParameters.desiredInstanceType ===
configurationItem.configuration.instanceType) {
        return 'COMPLIANT';
    }
        return 'NON_COMPLIANT';
```

   更新的代码

```
    if (configurationItem.resourceType !== 'AWS::XRay::EncryptionConfig') {
        return 'NOT_APPLICABLE';
    } else if (ruleParameters.desiredInstanceType ===
configurationItem.configuration.instanceType) {
        return 'COMPLIANT';
    }
```

```
        return 'NON_COMPLIANT';
```

6. 将以下内容添加到您的 IAM 的执行角色中以能够访问 X-Ray。这些权限允许对您的 X-Ray 资源进行只读访问。未能提供对相应资源的访问权限将导致在评估与规则关联的 Lambda 函数 AWS Config 时显示超出范围的消息。

```
{
    "Sid": "Stmt1529350291539",
    "Action": [
        "xray:GetEncryptionConfig"
    ],
    "Effect": "Allow",
    "Resource": "*"
 }
```

为 X 射线创建自定义 AWS Config 规则

创建 Lambda 函数后，记下该函数的 ARN，然后前往 AWS Config 控制台创建您的自定义规则。

为 X-Ray 创建 AWS Config 规则

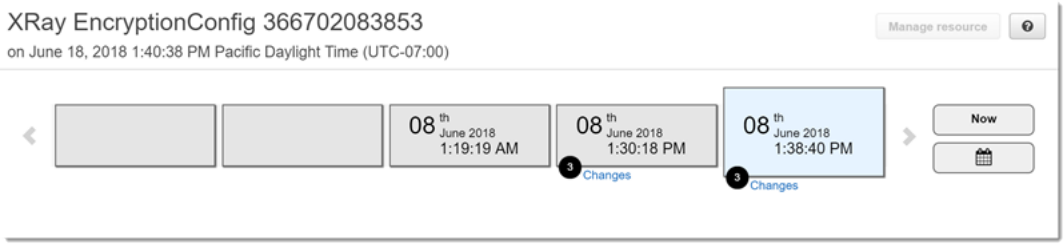
1. 打开 AWS Config 控制台的规则页面。
2. 选择添加规则，然后选择添加自定义规则。
3. 在 Function ARN（AWS Lambda 函数 ARN)中，插入您要使用的与 Lambda 函数关联的 ARN。
4. 选择要设置的触发器类型：
   - 配置更改 — 当任何与规则范围相匹配的资源在配置中发生变化时 AWS Config 触发评估。评估将在 AWS Config 发送配置项目变更通知后运行。
   - 定期- AWS Config 按您选择的频率对规则进行评估（例如，每 24 小时一次）。
5. 对于 Resource type (资源类型)，在 &xray; 部分选择EncryptionConfig在 X-Ray 部分中。
6. 选择保存。

AWS Config 控制台立即开始评估规则的合规性。完成评估可能需要几分钟时间。

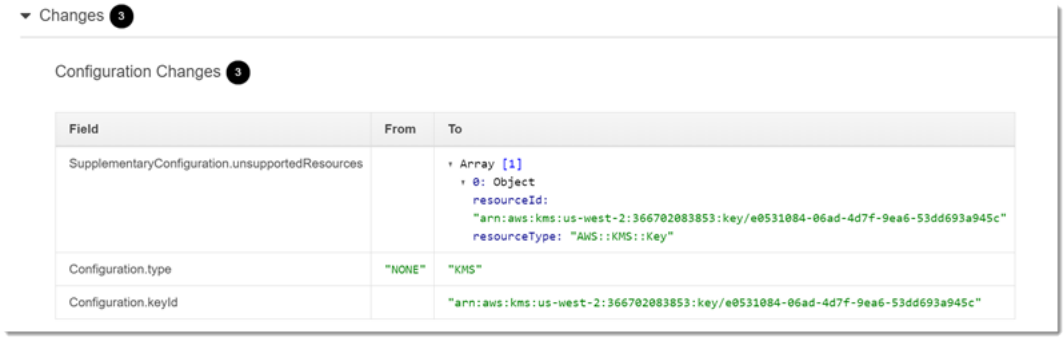
现在这条规则已经合规， AWS Config 可以开始编译审计历史记录了。 AWS Config 以时间表的形式记录资源变化。对于事件时间轴的每一次更改， AWS Config 都会生成一个 from/to 格式的表，以显示加密密钥的 JSON 表示形式发生了哪些变化。与之相关的两个字段更改 EncryptionConfig 是`Configuration.type`和`Configuration.keyID`。

示例结果

以下是显示在特定日期和 AWS Config 时间所做的更改的时间轴示例。



以下是 AWS Config 变更条目的示例。“从/更改为”格式阐明了有什么变化。此示例显示默认 X-Ray 加密设置改为定义的加密密钥。



Amazon SNS 通知

要收到配置更改的通知，请设置 AWS Config 为发布 Amazon SNS 通知。有关更多信息，请参阅通过电子邮件监控 AWS Config 资源更改。

Amazon 弹性计算云和 AWS X-Ray

您可以使用用户数据脚本在 Amazon EC2 实例上安装和运行 X-Ray 守护程序。有关说明，请参阅在亚马逊上运行 X-Ray 守护程序 EC2：

使用实例配置文件授予进程守护程序权限以上传跟踪数据到 X-Ray。有关更多信息，请参阅 授予进程守护程序向 X-Ray 发送数据的权限。

AWS Elastic Beanstalk 和 AWS X-Ray

AWS Elastic Beanstalk 平台包括 X-Ray 守护程序。您可以在 Elastic Beanstalk 控制台中设置选项或者使用配置文件运行进程守护程序。

在 Java SE 平台上，您可以使用 Buildfile 文件，通过 Maven 或 Gradle on-instance 构建应用程序。适用于 Java 适用于 Java 的 AWS SDK 的 X-Ray SDK 和，可从 Maven 获得，因此您只能部署应用程序代码并在实例上构建，从而避免捆绑和上传所有依赖项。

可以使用 Elastic Beanstalk 环境属性来配置 X-Ray 开发工具包。Elastic Beanstalk 用于将环境属性传递给应用程序的方法因平台而异。根据您的平台，使用 X-Ray 开发工具包的环境变量或系统属性。

- Node.js 平台 - 使用环境变量
- Java SE 平台 - 使用环境变量
- Tomcat 平台 - 使用系统属性

有关更多信息，请参阅《 AWS Elastic Beanstalk 开发人员指南》中的配置 AWS X-Ray 调试。

Elastic Load Balancin AWS X-Ray

弹性负载均衡应用程序负载均衡器将跟踪 ID 添加到传入 HTTP 请求的名为 `X-Amzn-Trace-Id` 的标头中。

```
X-Amzn-Trace-Id: Root=1-5759e988-bd862e3fe1be46a994272793
```

X-Ray 跟踪 ID 格式

X-Ray `trace_id` 由以连字符分隔的三组数字组成。例如，`1-58406520-a006649127e371903a2de979`。这包括：

- 版本号，即 `1`。
- 原始请求的时间，采用 Unix 纪元时间，为 8 个十六进制数字。

  例如，2016 年 12 月 1 日上午 10:00（太平洋标准时间）的纪元时间为 `1480615200` 秒，或者是十六进制数字 `58406520`。
- 跟踪的 96 位全局唯一标识符，使用 24 个十六进制数字。

负载均衡器不会将数据发送到 X-Ray，并且不会在您的服务地图中显示为节点。

有关更多信息，请参阅“弹性负载均衡开发人员指南”中的应用程序负载均衡器的请求跟踪。

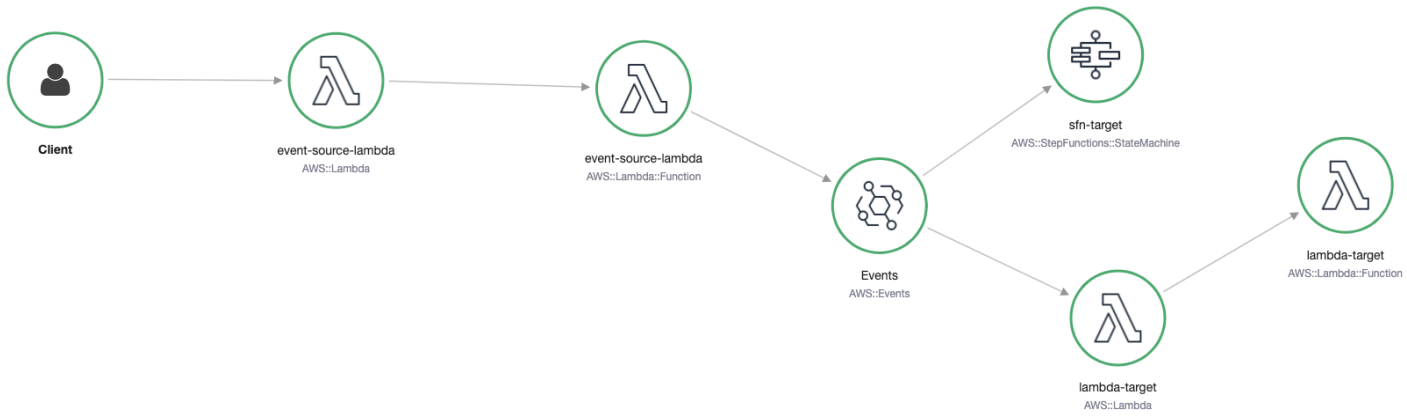
亚马逊 EventBridge 和 AWS X-Ray

AWS X-Ray 与 Amazon 集成 EventBridge 以跟踪通过的事件 EventBridge。如果使用 X-Ray SDK 进行检测的服务向发送事件 EventBridge，则跟踪上下文将传播到跟踪标头内的下游事件目标。X-Ray SDK 会自动获取跟踪标头并将其应用于任何后续检测。这种连续性使用户能够跟踪、分析和调试整个下游服务，并提供更完整的系统视图。

有关更多信息，请参阅《EventBridge 用户指南》中的 EventBridge X-Ray 集成。

在 X-Ray 服务映射上查看源和目标

X-Ray 跟踪地图显示连接源和目标服务 EventBridge 的事件节点，如下例所示：



将跟踪上下文传播到事件目标

X-Ray SDK 使 EventBridge 事件源能够将跟踪上下文传播到下游事件目标。以下特定于语言的示例演示了 EventBridge 从启用了主动跟踪的 Lambda 函数进行调用：

Java

为 X-Ray 添加必要的依赖项：

- AWS X-Ray 适用于 Java 的 SDK
- AWS X-Ray 适用于 Java 的录制器 SDK

```
package example;

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

```
import com.amazonaws.services.lambda.runtime.events.SQSEvent;
import com.amazonaws.xray.AWSXRay;
import com.amazonaws.services.eventbridge.AmazonEventBridge;
import com.amazonaws.services.eventbridge.AmazonEventBridgeClientBuilder;
import com.amazonaws.services.eventbridge.model.PutEventsRequest;
import com.amazonaws.services.eventbridge.model.PutEventsRequestEntry;
import com.amazonaws.services.eventbridge.model.PutEventsResult;
import com.amazonaws.services.eventbridge.model.PutEventsResultEntry;
import com.amazonaws.xray.handlers.TracingHandler;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.lang.StringBuilder;
import java.util.Map;
import java.util.List;
import java.util.Date;
import java.util.Collections;

/*
   Add the necessary dependencies for XRay:
   https://mvnrepository.com/artifact/com.amazonaws/aws-java-sdk-xray
   https://mvnrepository.com/artifact/com.amazonaws/aws-xray-recorder-sdk-aws-sdk
*/
public class Handler implements RequestHandler<SQSEvent, String>{
  private static final Logger logger = LoggerFactory.getLogger(Handler.class);

  /*
    build EventBridge client
  */
  private static final AmazonEventBridge eventsClient =
 AmazonEventBridgeClientBuilder
          .standard()
          // instrument the EventBridge client with the XRay Tracing Handler.
          // the AWSXRay globalRecorder will retrieve the tracing-context
          // from the lambda function and inject it into the HTTP header.
          // be sure to enable 'active tracing' on the lambda function.
          .withRequestHandlers(new TracingHandler(AWSXRay.getGlobalRecorder()))
          .build();

  @Override
  public String handleRequest(SQSEvent event, Context context)
  {
    PutEventsRequestEntry putEventsRequestEntry0 = new PutEventsRequestEntry();
```

```
    putEventsRequestEntry0.setTime(new Date());
    putEventsRequestEntry0.setSource("my-lambda-function");
    putEventsRequestEntry0.setDetailType("my-lambda-event");
    putEventsRequestEntry0.setDetail("{\"lambda-source\":\"sqs\"}");
    PutEventsRequest putEventsRequest = new PutEventsRequest();
    putEventsRequest.setEntries(Collections.singletonList(putEventsRequestEntry0));
    // send the event(s) to EventBridge
    PutEventsResult putEventsResult = eventsClient.putEvents(putEventsRequest);
    try {
      logger.info("Put Events Result: {}", putEventsResult);
    } catch(Exception e) {
      e.getStackTrace();
    }
    return "success";
  }
}
```

Python

将以下依赖项添加到 requirements.txt 文件中：

```
aws-xray-sdk==2.4.3
```

```
import boto3
from aws_xray_sdk.core import xray_recorder
from aws_xray_sdk.core import patch_all

# apply the XRay handler to all clients.
patch_all()

client = boto3.client('events')

def lambda_handler(event, context):
    response = client.put_events(
        Entries=[
            {
                'Source': 'foo',
                'DetailType': 'foo',
                'Detail': '{\"foo\": \"foo\"}'
            },
        ]
    )
```

```
    return response
```

Go

```
package main

import (
  "context"
  "github.com/aws/aws-lambda-go/lambda"
  "github.com/aws/aws-lambda-go/events"
  "github.com/aws/aws-sdk-go/aws/session"
  "github.com/aws/aws-xray-sdk-go/xray"
  "github.com/aws/aws-sdk-go/service/eventbridge"
  "fmt"
)

var client = eventbridge.New(session.New())


func main() {
 //Wrap the eventbridge client in the AWS XRay tracer
  xray.AWS(client.Client)
  lambda.Start(handleRequest)
}

func handleRequest(ctx context.Context, event events.SQSEvent) (string, error) {
  _, err := callEventBridge(ctx)
  if err != nil {
    return "ERROR", err
  }
  return "success", nil
}


func callEventBridge(ctx context.Context) (string, error) {
    entries := make([]*eventbridge.PutEventsRequestEntry, 1)
    detail := "{ \"foo\": \"foo\"}"
    detailType := "foo"
    source := "foo"
    entries[0] = &eventbridge.PutEventsRequestEntry{
        Detail: &detail,
        DetailType: &detailType,
        Source: &source,
```

```
    }

  input := &eventbridge.PutEventsInput{
     Entries: entries,
  }

  // Example sending a request using the PutEventsRequest method.
  resp, err := client.PutEventsWithContext(ctx, input)

  success := "yes"
  if err == nil { // resp is now filled
      success = "no"
      fmt.Println(resp)
  }
  return success, err
}
```

Node.js

```
const AWSXRay = require('aws-xray-sdk')
//Wrap the aws-sdk client in the AWS XRay tracer
const AWS = AWSXRay.captureAWS(require('aws-sdk'))
const eventBridge = new AWS.EventBridge()

exports.handler = async (event) => {

  let myDetail = { "name": "Alice" }

  const myEvent = {
    Entries: [{
      Detail: JSON.stringify({ myDetail }),
      DetailType: 'myDetailType',
      Source: 'myApplication',
      Time: new Date
    }]
  }

  // Send to EventBridge
  const result = await eventBridge.putEvents(myEvent).promise()

  // Log the result
  console.log('Result: ', JSON.stringify(result, null, 2))
```

```
}
```

C#

将以下 X-Ray 包添加到您的 C# 依赖项中：

```
<PackageReference Include="AWSXRayRecorder.Core" Version="2.6.2" />
<PackageReference Include="AWSXRayRecorder.Handlers.AwsSdk" Version="2.7.2" />
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Amazon;
using Amazon.Util;
using Amazon.Lambda;
using Amazon.Lambda.Model;
using Amazon.Lambda.Core;
using Amazon.EventBridge;
using Amazon.EventBridge.Model;
using Amazon.Lambda.SQSEvents;
using Amazon.XRay.Recorder.Core;
using Amazon.XRay.Recorder.Handlers.AwsSdk;
using Newtonsoft.Json;
using Newtonsoft.Json.Serialization;

[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.Json.JsonSerializer))]

namespace blankCsharp
{
  public class Function
  {
    private static AmazonEventBridgeClient eventClient;

    static Function() {
      initialize();
    }

    static async void initialize() {
      //Wrap the AWS SDK clients in the AWS XRay tracer
      AWSSDKHandler.RegisterXRayForAllServices();
      eventClient = new AmazonEventBridgeClient();
```

```
    }

    public async Task<PutEventsResponse> FunctionHandler(SQSEvent invocationEvent,
 ILambdaContext context)
    {
      PutEventsResponse response;
      try
      {
        response = await callEventBridge();
      }
      catch (AmazonLambdaException ex)
      {
        throw ex;
      }

      return response;
    }

    public static async Task<PutEventsResponse> callEventBridge()
    {
      var request = new PutEventsRequest();
      var entry = new PutEventsRequestEntry();
      entry.DetailType = "foo";
      entry.Source = "foo";
      entry.Detail = "{\"instance_id\":\"A\"}";
      List<PutEventsRequestEntry> entries = new List<PutEventsRequestEntry>();
      entries.Add(entry);
      request.Entries = entries;
      var response = await eventClient.PutEventsAsync(request);
      return response;
    }
  }
}
```

AWS Lambda 和 AWS X-Ray

您可以使用 AWS X-Ray 来跟踪您的 AWS Lambda 函数。Lambda 运行 X-Ray 进程守护程序并使用有关函数调用和运行该函数的详细信息记录分段。如需进一步检测，您可以将 X-Ray SDK 与您的函数绑定，以便记录传出调用以及添加注释和元数据。

如果您的 Lambda 函数由另一个已检测服务调用，则 Lambda 会跟踪已采样的请求，无需任何额外配置。上游服务可以是经过检测的 Web 应用程序或另一个 Lambda 函数。您的服务可以使用经过检测的 AWS SDK 客户端直接调用该函数，也可以使用经过检测的 HTTP 客户端调用 API Gateway API。

AWS X-Ray 支持使用 AWS Lambda 和 Amazon SQS 跟踪事件驱动的应用程序。使用 CloudWatch 控制台查看每个请求在 Amazon SQS 中排队并由下游 Lambda 函数处理的连接视图。来自上游消息生成者的跟踪会自动链接到来自下游 Lambda 使用者节点的跟踪，从而创建 end-to-end应用程序视图。有关更多信息，请参阅跟踪事件驱动型应用程序。

> **Note**
>
> 如果您为下游 Lambda 函数启用了跟踪，则还必须为调用下游函数的根 Lambda 函数启用跟踪，以便下游函数生成跟踪。

如果您的 Lambda 函数按计划运行，或者由未检测的服务调用，您可以将 Lambda 配置为通过活动跟踪采样和记录调用。

在 AWS Lambda 函数上配置 X-Ray 集成

1. 打开 AWS Lambda 管理控制台。
2. 从左侧导航栏中，选择函数。
3. 选择您的函数。
4. 在配置选项卡中，向下滚动到其他监控工具卡片。您也可以通过选择左侧导航窗格中的监控和操作工具来找到此卡片。
5. 选择编辑。
6. 在 AWS X-Ray 下，启用活动跟踪。

在运行时，通过对应的 X-Ray SDK，Lambda 也运行 X-Ray 进程守护程序。

Lambda SDKs 上的 X-Ray

- X-Ray SDK for G – Go 1.7 和更新版本的运行时
- X-Ray SDK for Java – Java 8 运行时
- X-Ray SDK for Node.js – Node.js 4.3 和更高版本的运行时
- X-Ray SDK for Python – Python 2.7、Python 3.6 和更新版本的运行时

- X-Ray SDK for .NET – .NET Core 2.0 和更新版本的运行时

要在 Lambda 上使用 X-Ray SDK，请在每次创建新版本时将其与您的函数代码绑定。您可以用检测运行在其他服务上的应用程序的相同方法来检测您的 Lambda 函数。主要差别在于您不使用 SDK 来检测传入请求、做出采样决策和创建分段。

检测 Lambda 函数和 Web 应用程序的另一个差别在于，Lambda 创建并发送到 X-Ray 的分段无法通过函数代码进行修改。您可以创建子分段并在其上记录注释和元数据，但无法将批注和元数据添加到父分段。

有关更多信息，请参阅 AWS Lambda 开发人员指南中的使用 AWS X-Ray。

亚马逊 SNS 和 AWS X-Ray

您可以使用 AWS X-Ray 亚马逊简单通知服务 (Amazon SNS) Simple Notification Service 来跟踪和分析通过您的 SNS 主题传送到您的 SNS 支持的订阅服务的请求。使用 X-Ray 跟踪与 Amazon SNS 分析消息及其后端服务的延迟（例如，请求在某个主题上花费了多长时间，以及将消息传送到该主题的每个订阅花费了多长时间）。Amazon SNS 对于标准主题和 FIFO 主题都支持 X-Ray 跟踪。

如果您从已经使用 X-Ray 检测过的服务发布到 Amazon SNS 主题，则 Amazon SNS 会将发布者的跟踪上下文传递给订阅用户。此外，还可以为从已检测 SNS 客户端发布的消息，打开活动跟踪将与 Amazon SNS 订阅相关的分段数据发送给 X-Ray。使用 Amazon SNS 控制台为某个 Amazon SNS 主题打开活动跟踪，或通过使用 Amazon SNS API 或 CLI。请参阅检测应用程序，详细了解如何检测 SNS 客户端。

配置 Amazon SNS 活动跟踪

您可以使用 Amazon SNS 控制台、CL AWS I 或 SDK 来配置 Amazon SNS 主动跟踪。

使用 Amazon SNS 控制台时，Amazon SNS 会尝试为 SNS 创建调用 X-Ray 所需的权限。如果您没有足够的权限修改 X-Ray 资源策略，则尝试可能会被拒绝。有关这些权限的更多信息，请参阅《Amazon Simple Notification Service 开发人员指南》中的 Amazon SNS 中的标识和访问管理和 Amazon SNS 访问控制示例。如需了解如何使用 Amazon SNS 控制台打开活动跟踪的更多信息，请参阅《Amazon Simple Notification Service 开发人员指南》中的在 Amazon SNS 主题上启用活动跟踪。

使用 AWS CLI 或 SDK 开启主动跟踪时，必须使用基于资源的策略手动配置权限。使用 `PutResourcePolicy` 基于资源的必要策略配置 X-Ray，以允许 Amazon SNS 将跟踪发送给 X-Ray。

Example Amazon SNS 主动跟踪的 X-Ray 基于资源的策略示例

以下示例策略文档指定了 Amazon SNS 将跟踪数据发送给 X-Ray 所需要的权限：

```
{
    Version: "2012-10-17",
    Statement: [
      {
        Sid: "SNSAccess",
        Effect: Allow,
        Principal: {
          Service: "sns.amazonaws.com",
        },
        Action: [
          "xray:PutTraceSegments",
          "xray:GetSamplingRules",
          "xray:GetSamplingTargets"
        ],
        Resource: "*",
        Condition: {
          StringEquals: {
            "aws:SourceAccount": "account-id"
          },
          StringLike: {
            "aws:SourceArn": "arn:partition:sns:region:account-id:topic-name"
          }
        }
      }
    ]
  }
```

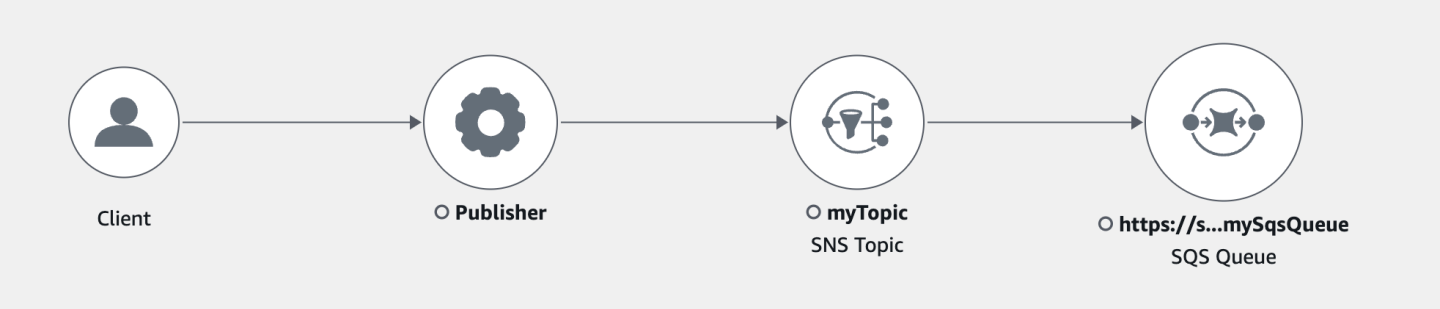
使用 CLI 创建基于资源的策略，赋予 Amazon SNS 将跟踪数据发送给 X-Ray 的权限：

```
aws xray put-resource-policy --policy-name MyResourcePolicy --policy-document
 '{ "Version": "2012-10-17", "Statement": [ { "Sid": "SNSAccess", "Effect": "Allow",
 "Principal": { "Service": "sns.amazonaws.com" }, "Action": [ "xray:PutTraceSegments",
 "xray:GetSamplingRules", "xray:GetSamplingTargets" ], "Resource": "*",
 "Condition": { "StringEquals": { "aws:SourceAccount": "account-id" }, "StringLike":
 { "aws:SourceArn": "arn:partition:sns:region:account-id:topic-name" } } } ] }'
```

要使用这些示例，请将*partition*、*regionaccount-id*、和*topic-name*替换为您的特定 AWS 分区、区域、账户 ID 和 Amazon SNS 主题名称。如需赋予所有 Amazon SNS 主题将跟踪数据发送给 X-Ray 的权限，请将主题名称替换为 *。

在 X-Ray 控制台中查看 Amazon SNS 发布者和订阅用户跟踪。

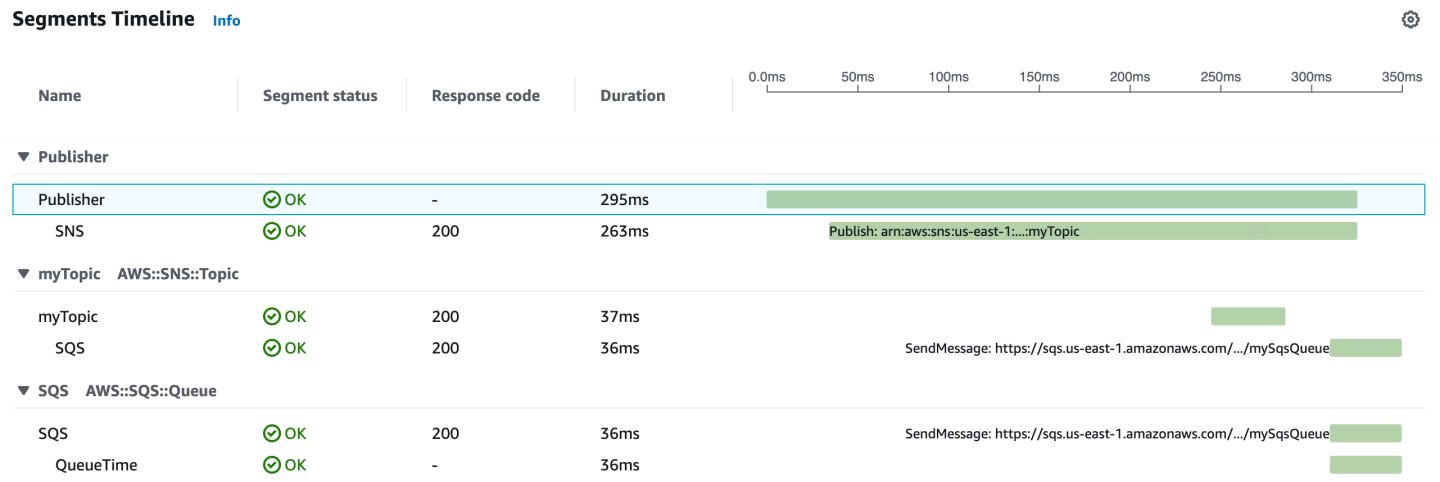
使用 X-Ray 控制台查看跟踪地图和跟踪详情，这些地图显示了 Amazon SNS 发布者和订阅用户的互联视图。为某个主题打开 Amazon SNS 活动跟踪后，X-Ray 跟踪地图和跟踪详情地图上会显示 Amazon SNS 发布者、Amazon SNS 主题和下游订阅用户的关联节点：



选择跨越 Amazon SNS 发布者和订阅用户的跟踪时，X-Ray 跟踪详情页面会显示跟踪详情地图和分段时间线。

Example 显示 Amazon SNS 发布者和订阅订阅用户的时间线示例

此示例显示的时间线中包含向某个 Amazon SNS 主题发送一条消息的 Amazon SNS 发布者，由 Amazon SNS 订阅用户处理。



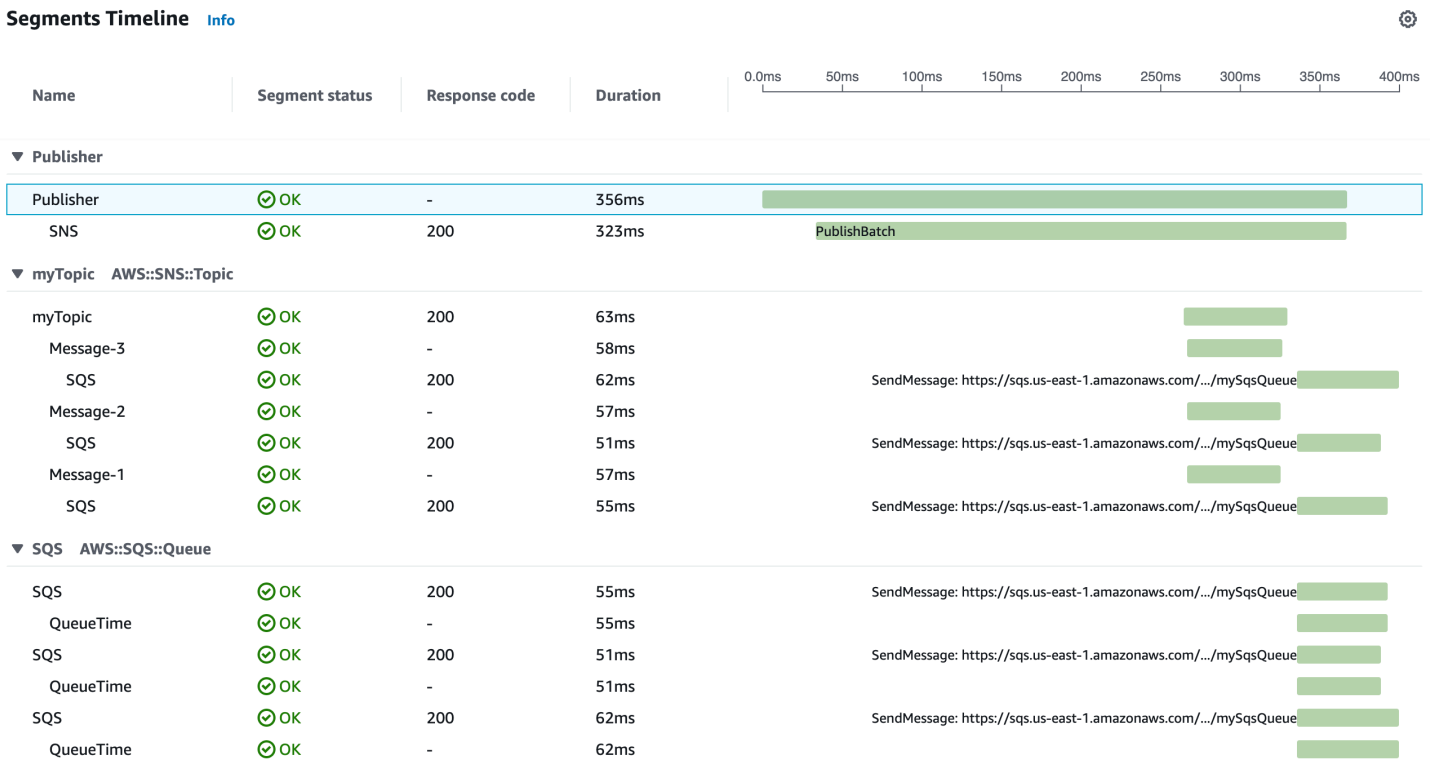
上面的示例时间线提供有关 Amazon SNS 消息流的详细信息：

- SNS 分段代表从客户端发出的 `Publish` API 调用的往返持续时间。

- myTopic 分段代表 Amazon SNS 响应发布请求的延迟。
- SQS 子分段表示 Amazon SNS 将消息发布到 Amazon SQS 队列所花费的往返时间。
- MyTopic 分段和 SQS 子分段之间的时间代表这条消息在 Amazon SNS 系统中花费的时间。

Example 包含批处理的 Amazon SNS 消息的时间线示例

如果在一个跟踪里批处理多条 Amazon SNS 消息，则分段时间线中会显示代表被处理的每条消息的分段。



AWS Step Functions 和 AWS X-Ray

AWS X-Ray 与集成 AWS Step Functions 以跟踪和分析 Step Functions 的请求。您可以可视化状态机的组件、确定性能瓶颈以及对导致错误的请求进行故障排除。有关更多信息，请参阅 AWS Step Functions 开发者指南中的AWS X-Ray 和 Step F unctions。

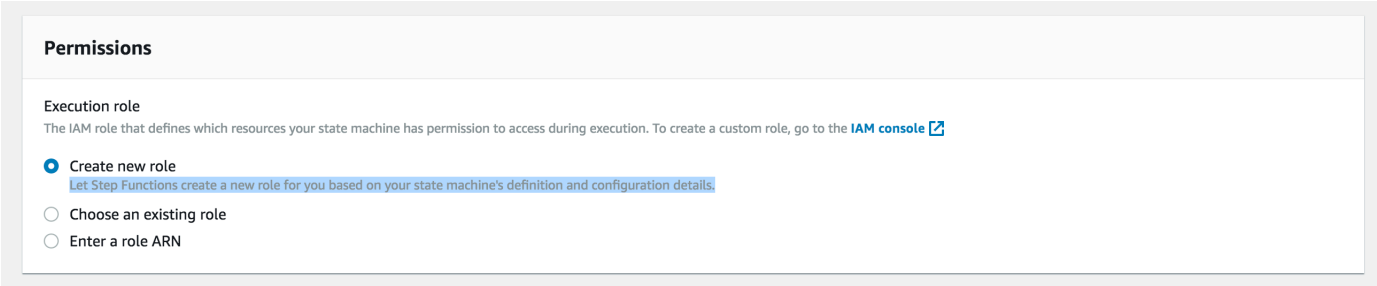
在创建新状态机时启用 X-Ray 跟踪

1. 打开 Step Functions 控制台，网址为https://console.aws.amazon.com/states/。
2. 选择创建状态机。

3. 在定义状态机页面，选择使用代码段创作或使用模板开始。如果选择运行示例项目，则无法在创建过程中启用 X-Ray 跟踪。相反地，请在创建状态机后启用 X-Ray 跟踪。
4. 选择下一步。
5. 在指定详细信息页面，配置状态机。
6. 选择启用 X-Ray 跟踪。

在现有状态机中启用 X-Ray 跟踪

1. 在 Step Functions 控制台中，选择要为其启用跟踪的状态机。
2. 选择编辑。
3. 选择启用 X-Ray 跟踪。
4. （可选）从“权限”窗口选择创建新角色，为状态机自动生成新角色以包含 X-Ray 权限。



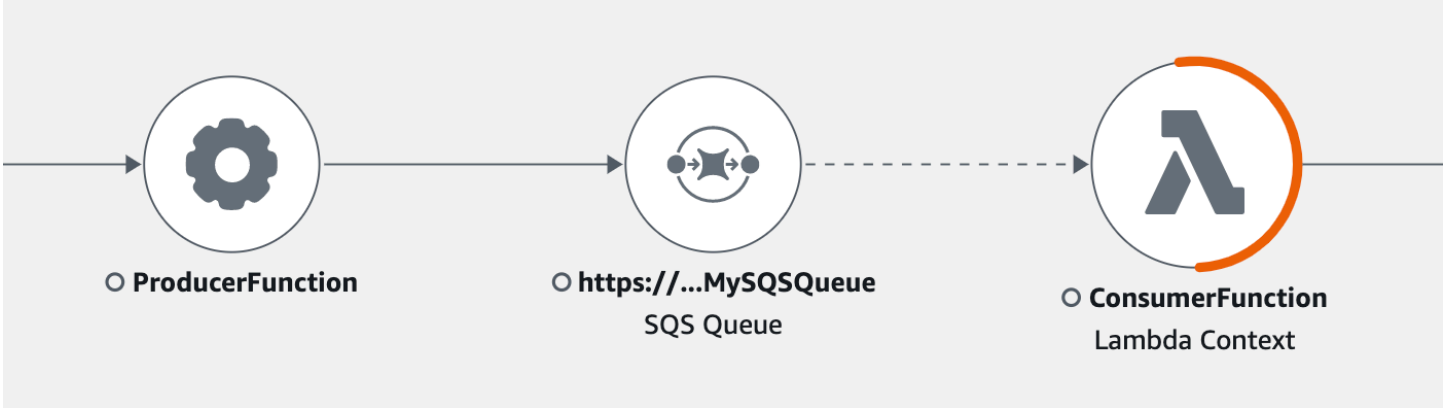
5. 选择保存。

> **Note**
>
> 当您创建新的状态机时，如果对请求进行了采样并在上游服务（例如 Amazon API Gateway 或 AWS Lambda）中启用了跟踪，则会自动对其进行跟踪。对于任何未通过控制台（例如通过 AWS CloudFormation 模板）配置的现有状态机，请检查您是否有授予启用 X-Ray 跟踪的足够权限的 IAM 策略。

亚马逊 SQS 和 AWS X-Ray

AWS X-Ray 与亚马逊简单队列服务 (Amazon SQS) Simple Queue Service 集成，可跟踪通过亚马逊 SQS 队列传递的消息。如果某项服务使用 X-Ray 开发工具包跟踪请求，则 Amazon SQS 可以发送跟踪标头并继续使用一致的跟踪 ID 将原始跟踪从发送者传播到使用者。跟踪连续性使用户能够跟踪、分析和调试整个下游服务。

AWS X-Ray 支持使用 Amazon SQS 和跟踪事件驱动的应用程序。 AWS Lambda使用 CloudWatch 控制台查看每个请求在 Amazon SQS 中排队并由下游 Lambda 函数处理的连接视图。来自上游消息生成者的跟踪会自动链接到来自下游 Lambda 使用者节点的跟踪，从而创建 end-to-end应用程序视图。有关更多信息，请参阅跟踪事件驱动型应用程序。



Amazon SQS 支持以下跟踪标头检测：

- 默认 HTTP 标头 — 当您通过 AWS 软件开发工具包调用亚马逊 SQS 时，X-Ray SDK 会自动将跟踪标头填充为 HTTP 标头。默认跟踪标头由 `X-Amzn-Trace-Id` 承载，对于包含在 `SendMessage` 或 `SendMessageBatch` 请求中的所有消息。请参阅 跟踪标头，详细了解有关默认 HTTP 标头的信息。
- **`AWSTraceHeader`** 系统属性 - `AWSTraceHeader` 是 Amazon SQS 保留的消息系统属性，用于承载队列中包含消息的 X-Ray 跟踪标头。即使无法通过 X-Ray SDK 进行自动检测时，也可以使用 `AWSTraceHeader`，例如在为新语言构建跟踪 SDK 时。如果同时设置了两个标头检测，则消息系统属性会覆盖 HTTP 跟踪标头。

在亚马逊上运行时 EC2，Amazon SQS 支持一次处理一条消息。这适用于在本地主机上运行以及使用容器服务（例如 Amazon ECS 或） AWS App Mesh。 AWS Fargate

Amazon SQS 消息大小和消息属性配额中都排除了跟踪标头。启用 X-Ray 跟踪不会超过您的 Amazon SQS 配额。要了解有关 AWS 配额的更多信息，请参阅 Amazon SQS 配额。

发送 HTTP 跟踪标头

Amazon SQS 中的发送者组件可以通过 `SendMessageBatch` 或 `SendMessage` 调用自动发送跟踪标头。在对 AWS SDK 客户端进行检测时，可以通过 X-Ray SDK 支持的所有语言自动跟踪这些客户端。在这些服务（例如，Amazon S3 存储桶或 Amazon SQS 队列）中追踪的资源 AWS 服务 和访问的资源在 X-Ray 控制台的跟踪地图上显示为下游节点。

要了解如何使用您的首选语言跟踪 AWS SDK 调用，请参阅支持的以下主题 SDKs：

- Go – 使用 X-Ray AWS SDK for Go 追踪 SDK 通话
- Java - 使用适用于 Java 的 X-Ray SD AWS K 追踪 SDK 调用
- Node.js – 使用适用于 Node.js 的 X-Ray SD AWS K 追踪 SDK 调用
- Python – 使用 AWS 适用于 Python 的 X-Ray 软件开发工具包追踪 SDK
- Ruby – 使用适用于 Ruby 的 X-Ray SD AWS K 追踪 SDK 调用
- .NET – 使用适用于.NET 的 X-Ray SD AWS K 追踪 SDK 调用

检索跟踪标头和恢复跟踪上下文

如果您使用的是 Lambda 下游使用器，则会自动传播跟踪上下文 要继续使用其他 Amazon SQS 使用器进行上下文传播，必须手动检测向接收方组件的交接。

恢复跟踪上下文主要分为以下三个步骤：

- 通过调用 `ReceiveMessage` API 从 `AWSTraceHeader` 属性的队列中接收消息。
- 从属性中检索跟踪标头。
- 从标头中恢复跟踪 ID。（可选）向分段添加更多指标。

下面是使用 X-Ray SDK for Java 编写的示例实施。

Example ：检索跟踪标头和恢复跟踪上下文

```
// Receive the message from the queue, specifying the "AWSTraceHeader"
ReceiveMessageRequest receiveMessageRequest = new ReceiveMessageRequest()
        .withQueueUrl(QUEUE_URL)
        .withAttributeNames("AWSTraceHeader");
List<Message> messages = sqs.receiveMessage(receiveMessageRequest).getMessages();

if (!messages.isEmpty()) {
    Message message = messages.get(0);

    // Retrieve the trace header from the AWSTraceHeader message system attribute
    String traceHeaderStr = message.getAttributes().get("AWSTraceHeader");
    if (traceHeaderStr != null) {
        TraceHeader traceHeader = TraceHeader.fromString(traceHeaderStr);

        // Recover the trace context from the trace header
```

```
        Segment segment = AWSXRay.getCurrentSegment();
        segment.setTraceId(traceHeader.getRootTraceId());
        segment.setParentId(traceHeader.getParentId());

 segment.setSampled(traceHeader.getSampled().equals(TraceHeader.SampleDecision.SAMPLED));
    }
}
```

亚马逊 S3 和 AWS X-Ray

AWS X-Ray 与 Amazon S3 集成，可跟踪更新应用程序的 S3 存储桶的上游请求。如果某项服务使用 X-Ray SDK 跟踪请求，则 Amazon S3 可以将跟踪标头发送给 AWS Lambda、Amazon SQS 和 Amazon SNS 等下游事件订阅用户。X-Ray 支持跟踪消息以实现 Amazon S3 事件通知。

您可以使用 X-Ray 跟踪地图查看 Amazon S3 与应用程序所用其他服务之间的连接。您还可以使用控制台查看指标，例如平均延迟和故障率。有关 X-Ray 控制台的更多信息，请参阅 使用 X-Ray 控制台。

Amazon S3 支持默认的 HTTP 标头检测。当您通过软件开发工具包调用 Amazon S3 时，X-Ray S AWS DK 会自动将跟踪标头填充为 HTTP 标头。默认跟踪标头由 `X-Amzn-Trace-Id` 承载。如需了解有关跟踪标头的更多信息，请参阅概念页面上的 跟踪标头。Amazon S3 跟踪上下文传播支持以下订阅用户：Lambda、SQS 和 SNS。由于 SQS 和 SNS 并不发送分段数据本身，因此，当被 S3 触发时不会显示在您的跟踪或跟踪地图中，即使它们会将跟踪标头传播给下游服务。

配置 Amazon S3 事件通知

通过 Amazon S3 通知功能，您可以在存储桶中发生某些事件时接收通知。然后，这些通知可以传播到应用程序中的以下目的地：

- Amazon Simple Notification Service (Amazon SNS)
- Amazon Simple Queue Service(Amazon SQS)
- AWS Lambda

有关受支持事件的列表，请参阅《Amazon Pinpoint 开发人员指南》中受支持的事件类型。

Amazon SNS 和 Amazon SQS

必须先授予 Amazon S3 权限，然后才能将通知发布到 SNS 主题或 SQS 队列。要授予这些权限，您需要将 AWS Identity and Access Management (IAM) 策略附加到目标 SNS 主题或 SQS 队列。如需了解有关所需 IAM 策略的更多信息，请参阅授予权限将消息发布到 SNS 主题或 SQS 队列。

有关将 SNS 和 SQS 与 X-Ray 集成的相关信息，请参阅 亚马逊 SNS 和 AWS X-Ray 和 亚马逊 SQS 和 AWS X-Ray。

AWS Lambda

使用 Amazon S3 控制台在 Amazon S3 存储桶上为 Lambda 函数配置事件通知时，控制台将在 Lambda 函数上设置必要的权限以便 Amazon S3 有权从存储桶调用函数。有关更多信息，请参阅《Amazon Simple Storage Service 控制台用户指南》中的如何为 S3 存储桶启用和配置事件通知？。

您也可以向 Amazon S3 授予调用您的 AWS Lambda Lambda 函数的权限。有关更多信息，请参阅 Lambda 开发者指南中的教程：将 AWS Lambda 与 Amazon S3 配合使用。 AWS

有关将 Lambda 与 X-Ray 集成的更多信息，请参阅在 Lambd a 中检测 Java 代码。 AWS

正在对您的应用程序进行检测 AWS X-Ray

检测应用程序涉及发送应用程序内传入和出站请求及其他事件的跟踪数据，以及与每个请求相关的元数据。您可以根据自身的特定需求，从多种不同检测选项中进行选择或结合使用：

- 自动检测 - 无需更改代码即可检测应用程序，常规方法包括更改配置、添加自动检测代理或使用其他机制。
- 库插入 — 对应用程序代码进行最少的更改，以添加针对特定库或框架（例如 AWS SDK、Apache HTTP 客户端或 SQL 客户端）的预建工具。
- 手动检测 - 在想要发送跟踪信息的每个位置，向应用程序添加检测代码。

有几个 SDKs、代理和工具可用于检测您的应用程序，以进行 X-Ray 跟踪。

主题

使用发行版对您的应用程序进行 AWS 检测 OpenTelemetry

OpenTelemetry (ADOT) AWS 发行版是基于云原生计算基金会 (CNCF) 项目的 AWS 发行版。OpenTelemetry OpenTelemetry 提供一组开源 APIs、库和代理，用于收集分布式跟踪和指标。该工具包是上游 OpenTelemetry 组件的发行版 SDKs，包括经过测试、优化、保护和支持的 AWS自动检测代理和收集器。

借助 ADOT，工程师只需对应用程序进行一次检测，即可将相关的指标和跟踪发送到多个 AWS 监控解决方案 CloudWatch AWS X-Ray，包括亚马逊和亚马逊 OpenSearch 服务。

将 X-Ray 与 ADOT 配合使用需要两个组件：启用 OpenTelemetry SDK 以与 X-Ray 配合使用，以及启用 Collecto OpenTelemetry r 的AWS Distro 以与 X-Ray 配合使用。有关将 AWS Distro OpenTelemetry 用于 with AWS X-Ray 和 other 的更多信息 AWS 服务，请参阅AWS 文档发行版。OpenTelemetry

有关语言支持和用法的更多信息，请参阅上的 O AWS bservability。 GitHub

> **Note**
>
> 现在，您可以使用 CloudWatch 代理从 Amazon EC2 实例和本地服务器收集指标、日志和跟踪。 CloudWatch 代理版本 1.300025.0 及更高版本可以从我们的 X-Ray 客户端收集痕迹 SDKs，然后将其发送到 OpenTelemetryX-R ay。使用 CloudWatch 代理代替 AWS Distro for OpenTelemetry (ADOT) Collector 或 X-Ray 守护程序来收集跟踪可以帮助您减少管理的代理数量。有关更多信息，请参阅《 CloudWatch 用户指南》中的CloudWatch 代理主题。

ADOT 包括以下内容：

- AWS Go 发行 OpenTelemetry 版
- AWS 适用于 Java 的 OpenTelemetry 发行版
- AWS 发行版适用于 OpenTelemetry JavaScript
- AWS 适用于 Python 的 OpenTelemetry 发行版
- AWS .NET 发行 OpenTelemetry 版

ADOT 目前包括适用于 Java 和 Python 的自动检测支持。此外，ADOT 还支持通过 ADOT 托管 Lambd AWS a 层使用 Java、Node.js 和 Python 运行时对 Lambda 函数及其下游请求进行自动检测。

SDKs 适用于 Java 和 Go 的 ADOT 支持 X-Ray 集中采样规则。如果您需要支持其他语言的 X-Ray 采样规则，请考虑使用 S AWS X-Ray DK。

> **Note**
>
> 你现在可以发送发送 W3C 跟踪 IDs 到 X-Ray。默认情况下，使用创建的跟踪 OpenTelemetry 具有基于 W3C 跟踪上下文规范的跟踪 ID 格式。这与使用 X-Ray SDK 或与 X-Ray 集成的 AWS 服务创建的跟踪 IDs 格式不同。为确保 X-Ray 接受 W3C 格式的跟踪 IDs ，您必须使用 AWS X-Ray Exporter 版本 0.86.0 或更高版本，该版本包含在 ADOT Collector 版本 0.34.0 及更高版本中。先前版本的导出器会验证跟踪 ID 时间戳，这可能会导致 W3C 跟踪被 IDs 拒绝。

使用以下方法对您的应用程序进行检测 AWS X-Ray SDKs

AWS X-Ray 包括一组特定于语言的工具， SDKs 用于对您的应用程序进行检测以向 X-Ray 发送跟踪。每个 X-Ray SDK 都提供以下内容：

- 拦截器，可添加到您的代码中以跟踪传入 HTTP 请求
- 客户端处理程序，用于检测 AWS SDK 客户端，您的应用程序使用这些客户端来调用其他客户端 AWS 服务
- HTTP 客户端，用于检测对其他内部和外部 HTTP Web 服务的调用

X-Ray SDKs 还支持对 SQL 数据库进行检测调用、自动 AWS SDK 客户端检测以及其他功能。该 SDK 不是直接将跟踪数据发送到 X-Ray，而是将 JSON 分段文档发送到侦听 UDP 流量的进程守护程序进程。X-Ray 进程守护程序将分段缓冲在队列中，并将分段批量上传到 X-Ray。

提供了以下特定于语言的内容 SDKs ：

- AWS X-Ray 适用于 Go 的 SDK
- AWS X-Ray 适用于 Java 的 SDK
- AWS X-Ray Node.js 的软件开发工具包
- AWS X-Ray Python 软件开发工具包
- AWS X-Ray 适用于 .NET 的 SDK
- AWS X-Ray 适用于 Ruby 的 SDK

X-Ray 目前包括适用于 Java 的自动检测支持。

在 AWS Distro for 和 X-Ray OpenTelemetry 之间进行选择 SDKs

X-Ray SDKs 随附的设备是提供的紧密集成的仪器解决方案的一部分 AWS。的 AWS 发行版 OpenTelemetry 是更广泛的行业解决方案的一部分，在该解决方案中，X-Ray 只是众多追踪解决方案之一。您可以使用任何一种方法在 X-Ray 中实现 end-to-end跟踪，但要确定最有用的方法，请务必了解其中的差异。

OpenTelemetry 如果您需要以下内容，我们建议您使用 AWS Distro 来测试您的应用程序：

- 无需重新检测代码即可将跟踪信息发送到多个不同的跟踪后端
- Support 支持每种语言的大量图书馆工具，由社区维护 OpenTelemetry
- 完全托管的 Lambda 层，使用 Java、Python 或 Node.js 时无需更改代码即可打包收集遥测数据所需的一切

> **Note**
>
> AWS Distro for 为你的 Lambda 函数 OpenTelemetry 提供了更简单的入门体验。但是，由于 OpenTelemetry 提供的灵活性，您的 Lambda 函数将需要额外的内存，并且调用可能会遇到冷启动延迟增加的情况，这可能会导致额外费用。如果您正在针对低延迟进行优化，并且不需要 OpenTelemetry的高级功能，例如可动态配置的后端目标，则可能需要使用 AWS X-Ray SDK 来检测您的应用程序。

如果您有以下需求，建议选择 X-Ray SDK 来检测应用程序：

- 紧密集成的单一供应商解决方案
- 与 X-Ray 集中采样规则集成，包括在 Node.js、Python、Ruby 或 .NET 时，能够从 X-Ray 控制台配置采样规则，以及跨多主机自动使用这些规则

Transaction Search

Transaction Search 是一种交互式分析体验，可供您用于全面了解您的应用程序事务跨度。跨度是分布式追踪中的基本操作单元，代表应用程序或系统中的特定操作或任务。每个跨度都会记录有关交易中特定部分的详细信息。这些详细信息包括开始和结束时间、持续时间以及相关的元数据，其中可以包括客户 IDs 和订单等业务属性 IDs。跨度按父子层次结构排列。这种层次结构形成了完整的跟踪，映射了跨不同组件或服务的交易流程。

有关更多信息，请参阅 Transaction Search。

OpenTelemetry 协议 (OTLP) 端点

OpenTelemetry 是一个开源可观测性框架，它为 IT 团队提供了用于收集和路由遥测数据的标准化协议和工具。该框架提供了一种统一的格式，用于检测、生成、收集应用程序遥测数据（例如指标、日志和跟踪），并将其导出到监测平台进行分析并获得洞察。通过使用 OpenTelemetry，团队可以避免供应商锁定，从而确保其可观察性解决方案的灵活性。

您可以使用直接 OpenTelemetry 向 OpenTelemetry 协议 (OTLP) 端点发送跟踪，并在应用程序信号中 CloudWatch 获得 out-of-the盒装应用程序性能监控体验。

有关更多信息，请参阅 OpenTelemetry。

使用 Go

有两种方法可用于检测 Go 应用程序，以将跟踪数据发送到 X-Ray：

- AWS Distro OpenTelemetry f or Go — 提供一组开源库的 AWS 发行版，用于通过 Distro for Collect AWS o r Collector 向包括亚马逊 CloudWatch和亚马逊 OpenSearch 服务在内的多个 AWS 监控解决方案发送相关的指标和跟踪。 AWS X-Ray OpenTelemetry
- AWS X-Ray SDK for Go — 一组库，用于通过 X-Ray 守护程序生成跟踪并将其发送到 X-Ray。

有关更多信息，请参阅 在 AWS Distro for 和 X-Ray OpenTelemetry 之间进行选择 SDKs。

AWS Go 发行 OpenTelemetry 版

使用 AWS Distr OpenTelemetry o for Go，您只需对应用程序进行一次检测，即可将相关的指标和跟踪发送到多个 AWS 监控解决方案 CloudWatch AWS X-Ray，包括亚马逊和亚马逊 OpenSearch 服务。将 X-Ray 与 AWS Distro 配合使用 OpenTelemetry 需要两个组件：启用 OpenTelemetry SDK 以与 X-Ray 配合使用，以及启用 Collecto OpenTelemetry r 的AWS Distro 以与 X-Ray 一起使用。

要开始使用，请参阅 AWS Distro for OpenTelemetry Go 文档。

有关将 Distro 用于 with AWS X-Ray 和 other 的 OpenTelemetry 更多信息 AWS 服务，请参阅 AWS Distro for OpenTelemetry 或 AWS Distro for AWS D ocuments。 OpenTelemetry

有关语言支持和用法的更多信息，请参阅上的 O AWS bservability。 GitHub

AWS X-Ray 适用于 Go 的 SDK

适用于 Go 的 X-Ray 开发工具包是一组面向 Go 应用程序的库，可提供类和方法来生成跟踪数据并将跟踪数据发送给 X-Ray 进程守护程序。跟踪数据包括有关应用程序处理的传入 HTTP 请求的信息，以及应用程序使用 AWS SDK、HTTP 客户端或 SQL 数据库连接器对下游服务进行的调用的信息。您还可以手动创建分段并在注释和元数据中添加调试信息。

使用以下命令从其GitHub存储库中下载 SDKgo get：

```
$ go get -u github.com/aws/aws-xray-sdk-go/...
```

对于 Web 应用程序，首先使用 `xray.Handler` 功能跟踪传入请求。消息处理程序为每个被跟踪的请求创建一个分段并在发送响应时完成该分段。当分段打开时，您可以使用开发工具包客户端的方法将信息添加到分段，并创建子分段以跟踪下游调用。开发工具包还会自动记录在分段打开时应用程序引发的异常。

对于由经过检测的应用程序或服务调用的 Lambda 函数，Lambda 会读取跟踪标头并自动跟踪采样的请求。对于其他函数，您可以将 Lambda 配置为采样和跟踪传入请求。无论哪种情况，Lambda 都会创建分段并将其提供给 X-Ray 开发工具包。

> **Note**
>
> 在 Lambda 上，X-Ray 开发工具包是可选的。如果您不在函数中使用它，您的服务映射仍将包含一个用于 Lambda 服务的节点，以及每个 Lambda 函数的节点。可通过添加该开发工具包检测函数代码，将子分段添加到 Lambda 记录的函数分段。请参阅AWS Lambda 和 AWS X-Ray了解更多信息。

接下来，使用对 AWS 函数的调用包装您的客户端。此步骤可确保 X-Ray 检测对任何客户端方法的调用。您还可以检测对 SQL 数据库的调用。

在开始使用开发工具包后，通过配置记录器和中间件来自定义其行为。您可以添加插件来记录有关应用程序上运行的计算资源的数据，通过定义采样规则来自定义采样行为，设置日志级别以在应用程序日志中查看来自开发工具包的更多或更少的信息。

记录有关请求以及应用程序在注释和元数据中所做的工作的其他信息。注释是简单的键值对，已为这些键值对编制索引以用于筛选条件表达式，以便您能够搜索包含特定数据的跟踪。元数据条目的限制性较低，并且可以记录整个对象和数组 - 可序列化为 JSON 的任何项目。

> **注释和元数据**
>
> 注释和元数据是您使用 X-Ray 开发工具包添加到分段的任意文本。系统会对注释编制索引，以便与筛选表达式一起使用。元数据未编制索引，但可以使用 X-Ray 控制台或 API 在原始分段中查看。您授予 X-Ray 读取权限的任何人都可以查看这些数据。

当代码中具有大量检测的客户端时，一个请求分段可包含大量子分段，检测的客户端发起的每个调用均对应一个子分段。您可以通过将客户端调用包含在自定义子分段中来整理子分段并为其分组。您可以为整个函数或任何代码部分创建自定义子分段，并记录子分段的元数据和注释，而不是编写父分段的所有内容。

要求

适用于 Go 的 X-Ray 开发工具包需要 Go 1.9 或更高版本。

该 SDK 在编译和运行时依赖于以下库：

- AWS 适用于 Go 的 SDK 版本 1.10.0 或更高版本

这些依赖项在开发工具包的 `README.md` 文件中声明。

参考文档

在下载开发工具包后，本地构建和托管文档以便在 Web 浏览器中查看文档。

查看参考文档

1. 导航到 `$GOPATH/src/github.com/aws/aws-xray-sdk-go`（Linux 或 Mac）目录或 `%GOPATH%\src\github.com\aws\aws-xray-sdk-go` (Windows) 文件夹
2. 运行 `godoc` 命令。

   ```
   $ godoc -http=:6060
   ```

3. 打开浏览器，定位到 `http://localhost:6060/pkg/github.com/aws/aws-xray-sdk-go/`。

配置适用于 Go 的 X-Ray 开发工具包

您可以通过环境变量、使用 `Configure` 对象调用 `Config` 或采用默认值，来为适用于 Go 的 X-Ray 开发工具包指定配置。环境变量优先于 `Config` 值，后者又优先于任何默认值。

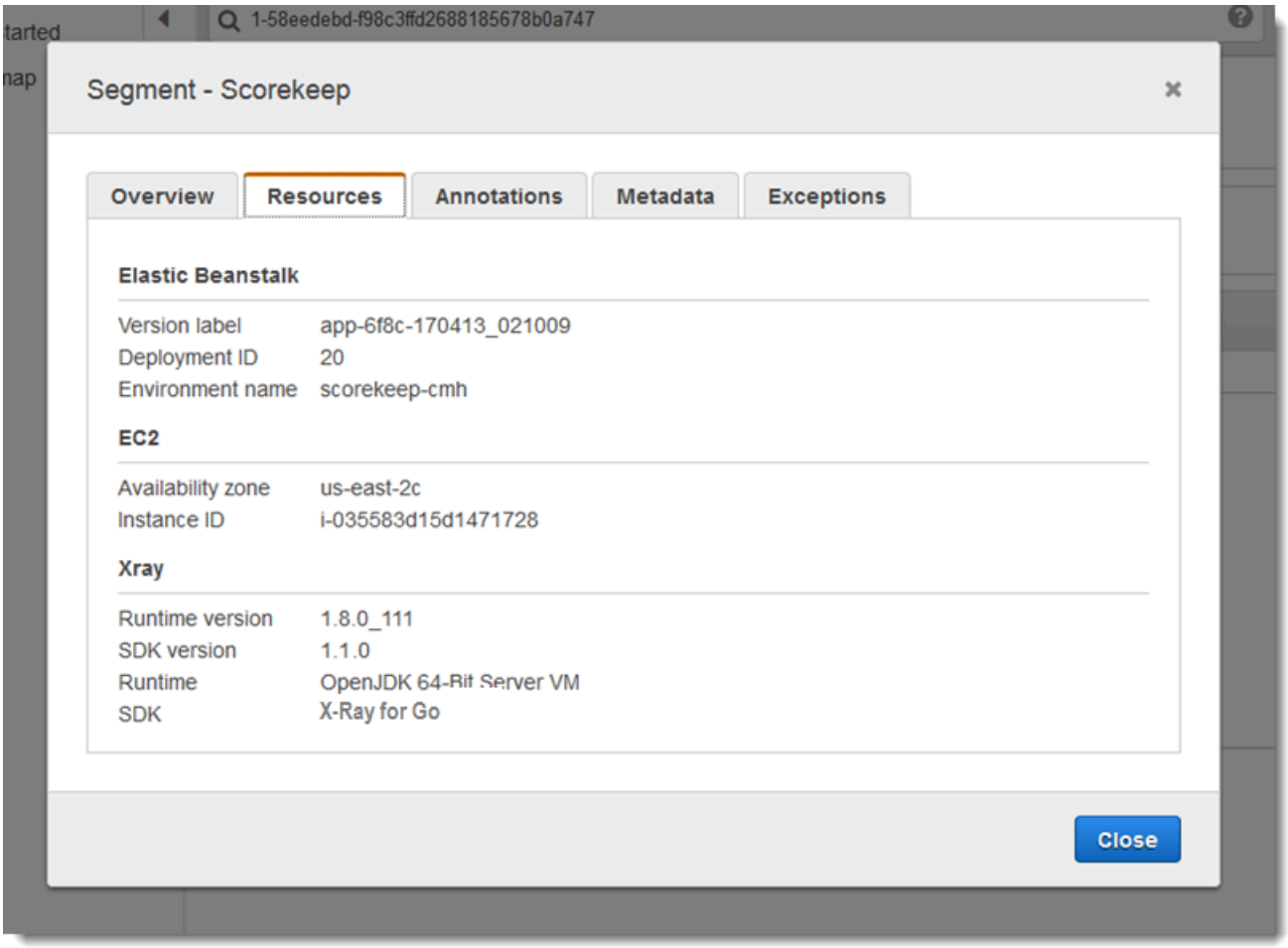
Sections

服务插件

`plugins` 用于记录有关托管应用程序的服务的信息。

插件

- Amazon EC2 — `EC2Plugin` 添加实例 ID、可用区和 CloudWatch 日志组。
- Elastic Beanstalk - `ElasticBeanstalkPlugin` 添加环境名称、版本标签和部署 ID。
- Amazon ECS — `ECSPlugin` 添加容器 ID。



要使用插件，请导入以下程序包之一。

```
"github.com/aws/aws-xray-sdk-go/awsplugins/ec2"
"github.com/aws/aws-xray-sdk-go/awsplugins/ecs"
"github.com/aws/aws-xray-sdk-go/awsplugins/beanstalk"
```

每个插件都有一个明确的 `Init()` 函数调用来加载插件。

Example ec2.Init()

```
import (
 "os"

 "github.com/aws/aws-xray-sdk-go/awsplugins/ec2"
 "github.com/aws/aws-xray-sdk-go/xray"
)

func init() {
  // conditionally load plugin
  if os.Getenv("ENVIRONMENT") == "production" {
    ec2.Init()
  }

  xray.Configure(xray.Config{
    ServiceVersion: "1.2.3",
  })
}
```

该 SDK 还使用插件设置为设置分段上的 `origin` 字段。这表示运行您的应用程序的 AWS 资源类型。当您使用多个插件时，SDK 使用以下解析顺序来确定来源： ElasticBeanstalk > EKS > ECS > EC2。

采样规则

该 SDK 使用您在 X-Ray 控制台中定义的采样规则来确定要记录的请求。默认规则跟踪每秒的第一个请求，以及所有将跟踪发送到 X-Ray 的服务的任何其他请求的百分之五。在 X-Ray 控制台中创建其他规则以自定义为每个应用程序记录的数据量。

该 SDK 按照定义的顺序应用自定义规则。如果请求与多个自定义规则匹配，则 SDK 仅应用第一条规则。

> **Note**
>
> 如果 SDK 无法访问 X-Ray 来获取采样规则，它将恢复为默认的本地规则，即每秒第一个请求以及每个主机所有其他请求的百分之五。如果主机无权调用采样，或者无法连接到 X-Ray 守护程序 APIs，后者充当 SDK 发出的 API 调用的 TCP 代理，则可能会发生这种情况。

您还可以将 SDK 配置为从 JSON 文档加载采样规则。在 X-Ray 采样不可用的情况下，SDK 可以使用本地规则作为备份，也可以只使用本地规则。

Example sampling-rules.json

```
{
  "version": 2,
  "rules": [
    {
      "description": "Player moves.",
      "host": "*",
      "http_method": "*",
      "url_path": "/api/move/*",
      "fixed_target": 0,
      "rate": 0.05
    }
  ],
  "default": {
    "fixed_target": 1,
    "rate": 0.1
  }
}
```

此示例定义了一个自定义规则和一个默认规则。自定义规则采用百分之五的采样率，对于 `/api/move/` 之下的路径要跟踪的请求数量不设下限。默认规则中每秒的第一个请求以及其他请求的百分之十。

在本地定义规则的缺点是，固定目标由记录器的每个实例独立应用而不是由 X-Ray 服务管理。随着您部署更多主机，固定速率会成倍增加，这使得控制记录的数据量变得更加困难。

开启后 AWS Lambda，您无法修改采样率。如果您的函数由检测服务调用，Lambda 将记录生成由该服务采样的请求的调用。如果启用了主动跟踪且不存在任何跟踪标头，则 Lambda 会做出采样决定。

要提供备份规则，请通过使用 `NewCentralizedStrategyWithFilePath` 指向本地采样 JSON 文件。

Example main.go - 本地采样规则

```
s, _ := sampling.NewCentralizedStrategyWithFilePath("sampling.json") // path to local
 sampling json
xray.Configure(xray.Config{SamplingStrategy: s})
```

要仅使用本地规则，请通过使用 NewLocalizedStrategyFromFilePath 指向本地采样 JSON 文件。

Example main.go - 禁用采样

```
s, _ := sampling.NewLocalizedStrategyFromFilePath("sampling.json") // path to local
 sampling json
xray.Configure(xray.Config{SamplingStrategy: s})
```

日志记录

> **Note**
>
> 从版本 1.0.0-rc.10 开始，xray.Config{} 字段 LogLevel 和 LogFormat 已弃用。

X-Ray 使用以下接口进行日志记录。默认记录器写入 stdout（LogLevelInfo 及跟高版本）。

```
type Logger interface {
 Log(level LogLevel, msg fmt.Stringer)
}

const (
 LogLevelDebug LogLevel = iota + 1
 LogLevelInfo
 LogLevelWarn
 LogLevelError
)
```

Example 写入 **io.Writer**

```
xray.SetLogger(xraylog.NewDefaultLogger(os.Stderr, xraylog.LogLevelError))
```

环境变量

您可以使用环境变量来配置适用于 Go 的 X-Ray 开发工具包。SDK 支持以下变量。

- AWS_XRAY_CONTEXT_MISSING - 设置为 RUNTIME_ERROR 在您的已检测代码尝试在分段未打开的情况下记录数据时引发异常。

有效值

- RUNTIME_ERROR— 引发运行时异常。
- LOG_ERROR— 记录错误并继续（默认）。
- IGNORE_ERROR— 忽略错误并继续。

对于在未打开任何请求时运行的启动代码或者会生成新线程的代码，如果您尝试在其中使用检测过的客户端，则可能发生与缺失分段或子分段相关的错误。

- AWS_XRAY_TRACING_NAME - 设置开发工具包用于分段的服务名称。
- AWS_XRAY_DAEMON_ADDRESS - 设置 X-Ray 进程守护程序侦听器的主机和端口。默认情况下，开发工具包会将跟踪数据发送到 127.0.0.1:2000。如果您已将进程守护程序配置为侦听不同端口或者进程守护程序在另一台主机上运行，则使用此变量。
- AWS_XRAY_CONTEXT_MISSING - 设置该值来确定开发工具包如何处理缺少上下文错误。对于在未打开任何请求时运行的启动代码或者会生成新线程的代码，如果您尝试在其中使用检测过的客户端，则可能发生与缺失分段或子分段相关的错误。
  - RUNTIME_ERROR - 默认情况下，开发工具包设置为抛出运行时异常。
  - LOG_ERROR - 记录错误并继续。

环境变量覆盖在代码中设置的等效值。

使用 Configure 方法

您还可以使用 Configure 方法配置适用于 Go 的 X-Ray 开发工具包。Configure 采用一个参数、一个 Config 对象以及以下可选字段。

DaemonAddr

此字符串指定 X-Ray 进程守护程序侦听器的主机和端口。如果未指定，X-Ray 将使用 AWS_XRAY_DAEMON_ADDRESS 环境变量的值。如果未设置该值，则将使用“127.0.0.1:2000”。

ServiceVersion

此字符串指定服务的版本。如果未指定，X-Ray 使用空字符串 ("")。

SamplingStrategy

此 SamplingStrategy 对象指定跟踪哪些应用程序调用。如果未指定，X-Ray 将使用 LocalizedSamplingStrategy，这将采用在 xray/resources/DefaultSamplingRules.json 中定义的策略。

StreamingStrategy

此StreamingStrategy对象指定在RequiresStreaming返回 true 时是否流式传输片段。如果未指定，X-Ray 将使用 DefaultStreamingStrategy，这将在子分段数超过 20 个时流式传输采样分段。

ExceptionFormattingStrategy

此 ExceptionFormattingStrategy 对象指定您希望如何处理各种异常。如果未指定，X-Ray 将使用带有 DefaultExceptionFormattingStrategy 类型的 XrayError、错误消息和堆栈跟踪的 error。

使用适用于 Go 的 X-Ray 开发工具包检测传入 HTTP 请求

您可以使用 X-Ray SDK 来跟踪您的应用程序在亚马逊或亚马逊 EC2 ECS 中的 EC2 实例上提供的传入 HTTP 请求。 AWS Elastic Beanstalk

使用 xray.Handler 检测传入 HTTP 请求。适用于 Go 的 X-Ray 开发工具包在 http.Handler 类中实现标准的 Go 库 xray.Handler 接口以截取 Web 请求。xray.Handler 类通过 http.Handler 包装提供的 xray.Capture（使用请求的上下文，解析传入的标头，根据需要添加响应标头），并设置特定于 HTTP 的跟踪字段。

当您使用此类来处理 HTTP 请求和响应时，适用于 Go 的 X-Ray 开发工具包将为每个采样请求创建一个分段。此分段包括 HTTP 请求的计时、方法和处置。其他检测会在此分段上创建子分段。

> **Note**
>
> 对于 AWS Lambda 函数，Lambda 会为每个采样请求创建一个分段。请参阅AWS Lambda 和 AWS X-Ray了解更多信息。

以下示例在端口 8000 上截取请求并返回“Hello!” 作为响应。它通过任何应用程序创建分段 myApp 并检测调用。

Example main.go

```
func main() {
  http.Handle("/", xray.Handler(xray.NewFixedSegmentNamer("MyApp"),
 http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
    w.Write([]byte("Hello!"))
```

```
  }))

  http.ListenAndServe(":8000", nil)
}
```

每个分段都有一个名称，用于在服务映射中标识您的应用程序。可以静态命名分段，也可以将 SDK 配置为根据传入请求中的主机标头对其进行动态命名。动态命名允许根据请求中的域名对跟踪进行分组，并且在名称不匹配预期模式时（例如，如果主机标头是伪造的）应用默认名称。

> **转发的请求**
>
> 如果负载均衡器或其他中间将请求转发到您的应用程序，X-Ray 会提取请求 `X-Forwarded-For` 标头中的客户端 IP 而非 IP 数据包中的源 IP。由于转发的请求记录的客户端 IP 可以伪造，因此不应信任。

在转发请求时，SDK 在分段中设置附加字段来指示此行为。如果分段包含设置为 `x_forwarded_for` 的字段 `true`，则从 HTTP 请求的 `X-Forwarded-For` 标头获取客户端 IP。

处理程序使用包含以下信息的 `http` 块为每个传入请求创建一个分段：

- HTTP 方法 - GET、POST、PUT、DELETE 等。
- 客户端地址 - 发送请求的客户端的 IP 地址。
- 响应代码 - 已完成请求的 HTTP 响应代码。
- 时间 - 开始时间（收到请求时）和结束时间（发送响应时）。
- 用户代理 - 请求中的 `user-agent`。
- 内容长度 - 响应中的 `content-length`。

配置分段命名策略

AWS X-Ray 使用服务名称来标识您的应用程序，并将其与您的应用程序使用的其他应用程序、数据库 APIs、外部数据库和 AWS 资源区分开来。当 X-Ray SDK 为传入请求生成分段时，会将应用程序的服务名称记录在分段的名称字段中。

X-Ray SDK 可以用在 HTTP 请求标头中的 hostname 来命名分段。不过，此标头可以伪造，会导致服务地图中出现意料之外的节点。为防止 SDK 由于包含伪造的主机标头的请求而错误地命名分段，必须为传入请求指定一个默认名称。

如果应用程序为多个域的请求提供服务，则可以将 SDK 配置为使用动态命名策略以在分段名称中反映出这一点。动态命名策略允许 SDK 将主机名用于符合预期模式的请求，并将默认名称应用于不符合预期模式的请求。

例如，可能有一款应用程序为发送到三个子域的请求提供服务，分别为 `www.example.com`、`api.example.com` 和 `static.example.com`。可以使用格式 `*.example.com` 的动态命名策略以识别包含不同名称的子域的分段，服务地图上因此会显示三个服务节点。如果应用程序收到包含与该格式不匹配的 hostname 的请求，您将会在服务地图上看到第四个节点，以及您指定的回退名称。

要对所有请求分段使用相同名称，请在创建处理程序时指定应用程序的名称，如前面的部分中所示。

> **Note**
>
> 您可以使用 `AWS_XRAY_TRACING_NAME` 环境变量覆盖您在代码中定义的默认服务名称。

动态命名策略定义一个主机名应匹配的模式和一个在 HTTP 请求中的主机名与该模式不匹配时要使用的默认名称。要动态命名分段，请使用 `NewDynamicSegmentNamer` 配置默认名称和要匹配的模式。

Example main.go

如果请求中的主机名与模式 `*.example.com` 匹配，请使用主机名。否则，请使用 `MyApp`。

```
func main() {
  http.Handle("/", xray.Handler(xray.NewDynamicSegmentNamer("MyApp", "*.example.com"),
 http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
    w.Write([]byte("Hello!"))
  })))

  http.ListenAndServe(":8000", nil)
}
```

使用 X-Ray AWS SDK for Go 追踪 SDK 通话

当您的应用调用 AWS 服务 以存储数据、写入队列或发送通知时，X-Ray SDK for Go 会按子分段跟踪下游的调用。在这些服务（例如，Amazon S3 存储桶或 Amazon SQS 队列）中追踪的资源 AWS 服务 和访问的资源在 X-Ray 控制台的跟踪地图上显示为下游节点。

要跟踪 AWS SDK 客户端，请将客户端对象与 `xray.AWS()` 调用一起包装，如以下示例所示。

Example main.go

```
var dynamo *dynamodb.DynamoDB
func main() {
  dynamo = dynamodb.New(session.Must(session.NewSession()))
  xray.AWS(dynamo.Client)
}
```

然后，当您使用 AWS SDK 客户端时，使用调用方法的 `withContext` 版本，在 `context` 中将其从 `http.Request` 对象传递到处理程序。

Example main.go — AWS SDK 调用

```
func listTablesWithContext(ctx context.Context) {
  output := dynamo.ListTablesWithContext(ctx, &dynamodb.ListTablesInput{})
  doSomething(output)
}
```

对于所有服务，都可以在 X-Ray 控制台中看到调用的 API 的名称。X-Ray 开发工具包会为一部分服务将信息添加到分段，从而在服务地图中提供更高的粒度。

例如，当使用经过检测的 DynamoDB 客户端发出调用时，对于针对表的调用，开发工具包会将表名称添加到分段中。在控制台中，每个表在服务地图中显示为一个独立的节点，以及没有表作为目标的调用的一般 DynamoDB 节点。

Example 对 DynamoDB 进行调用以保存项目的子分段

```
{
  "id": "24756640c0d0978a",
  "start_time": 1.480305974194E9,
  "end_time": 1.4803059742E9,
  "name": "DynamoDB",
  "namespace": "aws",
  "http": {
    "response": {
      "content_length": 60,
      "status": 200
    }
  },
  "aws": {
    "table_name": "scorekeep-user",
```

```
    "operation": "UpdateItem",
    "request_id": "UBQNSO5AEM8T4FDA4RQDEB94OVTDRVV4K4HIRGVJF66Q9ASUAAJG",
  }
}
```

在您访问指定的资源时，对以下服务的调用会在服务地图中创建额外的节点。没有定向到特定资源的调用，为服务创建了通用节点。

- Amazon DynamoDB - 表名称
- Amazon Simple Storage Service - 存储桶和键名称
- Amazon Simple Queue Service - 队列名称

使用适用于 Go 的 X-Ray 开发工具包跟踪对下游 HTTP Web 服务的调用

当您的应用程序调用微服务或公共 HTTP 时 APIs，您可以使用`xray.Client`将这些调用视为 Go 应用程序的子段，如以下示例所示，其中 http-client 是一个 HTTP 客户端。

客户端创建所提供的 HTTP 客户端的阴影副本，默认为 `http.DefaultClient`，并带有使用 `xray.RoundTripper` 包装的往返处理器。

Example

<caption>main.go - HTTP 客户端</caption>

```
myClient := xray.Client(http-client)
```

<caption>main.go — 使用 ctxhttp 库跟踪下游 HTTP 调用</caption>

以下示例借助使用 `xray.Client` 的 ctxhttp 库来检测传出 HTTP 调用。可以传递来自上游调用的 `ctx`。这样可以确保使用现有的区段上下文。例如，X-Ray 不允许在 Lambda 函数中创建新的分段，因此应使用现有的 Lambda 分段上下文。

```
resp, err := ctxhttp.Get(ctx, xray.Client(nil), url)
```

使用适用于 Go 的 X-Ray 开发工具包跟踪 SQL 查询

要跟踪对 PostgreSQL 或 MySQL 的 SQL 调用，请将 `sql.Open` 调用替换为 `xray.SQLContext`，如以下示例所示。如果可能，请使用 URLs 而不是配置字符串。

Example main.go

```
func main() {
  db, err := xray.SQLContext("postgres", "postgres://user:password@host:port/db")
  row, err := db.QueryRowContext(ctx, "SELECT 1") // Use as normal
}
```

使用适用于 Go 的 X-Ray 开发工具包生成自定义子分段

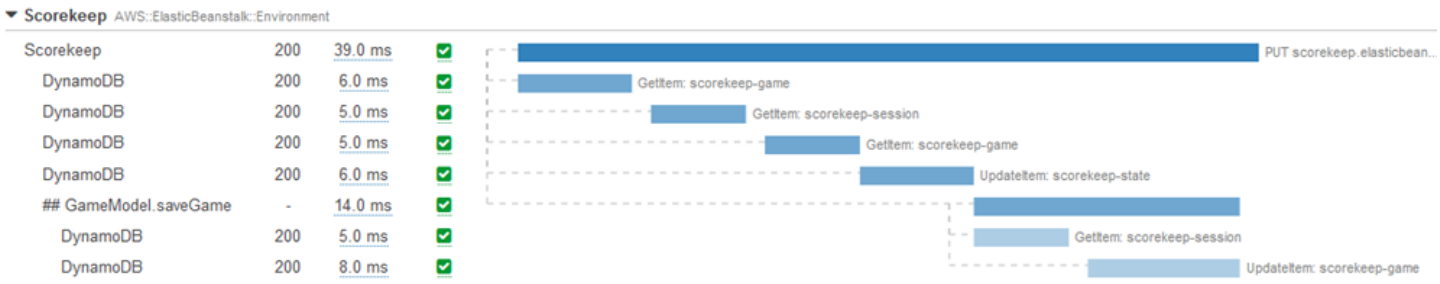
子分段可为跟踪的分段扩展为了给请求提供服务而已完成的工作的详细信息。每次使用已检测的客户端进行调用时，X-Ray SDK 在子分段中记录生成的信息。您可以创建其他子分段来分组其他子分段，来度量某个代码段的性能如何，或是来记录注释和元数据。

使用 `Capture` 方法创建有关函数的子分段。

Example main.go - 自定义子分段

```
func criticalSection(ctx context.Context) {
  //this is an example of a subsegment
  xray.Capture(ctx, "GameModel.saveGame", func(ctx1 context.Context) error {
    var err error

    section.Lock()
    result := someLockedResource.Go()
    section.Unlock()

    xray.AddMetadata(ctx1, "ResourceResult", result)
  })
```

以下屏幕截图中显示的示例说明了 `saveGame` 子分段如何显示在应用程序 `Scorekeep` 的跟踪中。



使用 X-Ray SDK for Go，将注释和元数据添加到分段

可以利用注释和元数据记录与请求、环境或应用程序相关的其他信息。可以将注释和元数据添加到 X-Ray 开发工具包创建的分段或您创建的自定义子分段。

注释是带字符串、数字或布尔值的键值对。系统会对注释编制索引，以便与筛选表达式一起使用。使用注释记录要用于对控制台中的跟踪进行分组的数据或在调用 `GetTraceSummaries` API 时使用的数据。

元数据是可以具有任何类型值的键-值对，包括对象和列表，但没有编制索引，无法与筛选条件表达式一起使用。使用元数据记录要存储在跟踪中但不需要用于搜索跟踪的其他数据。

除了注释和元数据之外，您还可以在分段上记录用户 ID 字符串。用户 IDs 被记录在区段的单独字段中，并编制索引以供搜索使用。

Sections

使用 X-Ray SDK for Go 记录注释

使用注释记录有关要为其编制索引以进行搜索的分段的信息。

注释要求

- 键 - X-Ray 注释的键最多可以包含 500 个字母数字字符。除了点或句点（.）之外，不能使用空格或符号
- 值 - X-Ray 注释的值最多可以包含 1,000 个 Unicode 字符。
- 注释的数量 - 每个跟踪最多可使用 50 条注释。

要记录注释，请使用一个包含您要与分段关联的元数据的字符串来调用 `AddAnnotation`。

```
xray.AddAnnotation(key string, value interface{})
```

开发工具包将注释以键-值对的形式记录在分段文档的 `annotations` 对象中。使用相同键调用两次 `AddAnnotation` 将覆盖同一分段上之前记录的值。

要查找具有带特定值的注释的跟踪，请在annotation[*key*]筛选表达式中使用 关键字。

使用 X-Ray SDK for Go 记录元数据

使用元数据记录有关您无需为其编制索引以进行搜索的分段的信息。

要记录元数据，请使用一个包含您要与分段关联的元数据的字符串来调用 AddMetadata。

```
xray.AddMetadata(key string, value interface{})
```

IDs 使用 X-Ray SDK for Go 录制用户

记录请求细分中的用户，以识别发送请求的用户。 IDs

要记录用户 IDs

1. 从 AWSXRay 获取对当前分段的引用。

   ```
   import (
     "context"
     "github.com/aws/aws-xray-sdk-go/xray"
   )

   mySegment := xray.GetSegment(context)
   ```

2. 使用发送请求的用户的字符串 ID 调用 setUser。

   ```
   mySegment.User = "U12345"
   ```

要查找用户 ID 的跟踪，请在user筛选表达式中使用 关键字。

使用 Java

有两种方法可用于检测 Java 应用程序，以将跟踪数据发送到 X-Ray：

- AWS OpenTelemetry Java 版 Distro — 提供一组开源库的 AWS 发行版，用于通过 Distro for Collect AWS o r Collector 向多个 AWS 监控解决方案（包括亚马逊和亚马逊 OpenSearch 服务）发送相关的指标和跟踪。 CloudWatch AWS X-Ray OpenTelemetry
- AWS X-Ray 适用于 Java 的 SDK — 一组库，用于通过 X-Ray 守护程序生成跟踪并将其发送到 X-Ray。

有关更多信息，请参阅 在 AWS Distro for 和 X-Ray OpenTelemetry 之间进行选择 SDKs。

AWS 适用于 Java 的 OpenTelemetry 发行版

使用适用于 OpenTelemetry (ADOT) Java 的 AWS Distro，您可以对应用程序进行一次检测，然后将相关的指标和跟踪发送到多个 AWS 监控解决方案 CloudWatch，包括亚马逊 AWS X-Ray和亚马逊服务。 OpenSearch 将 X-Ray 与 ADOT 配合使用需要两个组件：启用 OpenTelemetry SDK 以与 X-Ray 配合使用，以及启用 Collecto OpenTelemetryr 的AWS Distro 以与 X-Ray 配合使用。ADOT Java 支持自动检测，使您的应用程序无需更改代码即可发送跟踪信息。

要开始使用，请参阅 OpenTelemetry Java AWS 发行版文档。

有关将 Distro 用于 with AWS X-Ray 和 other 的 OpenTelemetry 更多信息 AWS 服务，请参阅 AWS Distro for OpenTelemetry 或 AWS Distro for AWS D ocuments。 OpenTelemetry

有关语言支持和用法的更多信息，请参阅上的 O AWS bservability。 GitHub

AWS X-Ray 适用于 Java 的 SDK

X-Ray SDK for Java 是一组面向 Java Web 应用程序的库，可提供类和方法来生成跟踪数据并将跟踪数据发送给 X-Ray 进程守护程序。跟踪数据包括有关应用程序处理的传入 HTTP 请求的信息，以及应用程序使用 AWS SDK、HTTP 客户端或 SQL 数据库连接器对下游服务进行的调用的信息。您还可以手动创建分段并在注释和元数据中添加调试信息。

X-Ray SDK for Java 是一个开源项目。你可以关注该项目并在 github 上 GitHub提交议题和拉取请求。com/aws/aws-xray-sdk-java

首先通过添加AWSXRayServletFilter 作为 servlet 筛选器来跟踪传入请求。servlet 筛选器会创建分段。当分段打开时，您可以使用开发工具包客户端的方法将信息添加到分段，并创建子分段以跟踪下游调用。开发工具包还会自动记录在分段打开时应用程序引发的异常。

从版本 1.3 开始，您可以使用 Spring 中的面向方面的编程 (AOP) 来检测应用程序。这意味着你可以在应用程序运行时对其进行检测，而无需在 AWS应用程序的运行时中添加任何代码。

接下来，使用适用于 Java 的 X-Ray SDK，在构建配置中包含 SDK Instrumentor 子模块，对 适用于 Java 的 AWS SDK 客户进行检测。每当您使用已检测的客户端调用下游 AWS 服务 或资源时，SDK 都会在子分段中记录有关该调用的信息。 AWS 服务 您在服务中访问的资源将作为下游节点显示在跟踪地图上，以帮助您识别各个连接上的错误和限制问题。

如果您不想检测所有下游调用 AWS 服务，则可以省略 Instrumentor 子模块，然后选择要检测的客户端。通过向 AWS SDK 服务客户端添加TracingHandler来检测各个客户端。

其他 X-Ray SDK for Java 子模块为对 HTTP Web APIs 和 SQL 数据库的下游调用提供了工具。您可以使用 X-Ray SDK for Java 的 HTTPClient 版本和 Apache HTTP 子模块中的 HTTPClientBuilder 来检测 Apache HTTP 客户端。要检测 SQL 查询，请将 SDK 的拦截程序添加到数据源。

在开始使用 SDK 后，通过配置记录器和 servlet 筛选器来自定义其行为。您可以添加插件来记录有关应用程序上运行的计算资源的数据，通过定义采样规则来自定义采样行为，设置日志级别以在应用程序日志中查看来自开发工具包的更多或更少的信息。

记录有关请求以及应用程序在注释和元数据中所做的工作的其他信息。注释是简单的键值对，已为这些键值对编制索引以用于筛选条件表达式，以便您能够搜索包含特定数据的跟踪。元数据条目的限制性较低，并且可以记录整个对象和数组 - 可序列化为 JSON 的任何项目。

> **注释和元数据**
>
> 注释和元数据是您使用 X-Ray 开发工具包添加到分段的任意文本。系统会对注释编制索引，以便与筛选表达式一起使用。元数据未编制索引，但可以使用 X-Ray 控制台或 API 在原始分段中查看。您授予 X-Ray 读取权限的任何人都可以查看这些数据。

当代码中具有大量检测的客户端时，一个请求分段可包含许多子分段，检测的客户端发起的每个调用均对应一个子分段。您可以通过将客户端调用包含在自定义子分段中来整理子分段并为其分组。您可以为整个函数或任何代码部分创建自定义子分段，并记录子分段的元数据和注释，而不是编写父分段的所有内容。

子模块

您可以从 Maven 下载 X-Ray SDK for Java。X-Ray SDK for Java 按使用案例被拆分为子模块，其中的材料清单用于版本管理：

- aws-xray-recorder-sdk-core（必需）- 用于创建分段和传输分段的基本功能。包括 AWSXRayServletFilter 用于检测传入请求。
- aws-xray-recorder-sdk-aws-sdk— 通过添加跟踪 适用于 Java 的 AWS SDK 客户端作为请求处理程序，对客户端发 AWS 服务 出的调用进行监测。
- aws-xray-recorder-sdk-aws-sdk-v2— 通过添加跟踪客户端作为请求拦截器，对 AWS 服务使用 适用于 Java 的 AWS SDK 2.2 及更高版本的客户端进行的调用进行监测。
- aws-xray-recorder-sdk-aws-sdk-instrumentor— 使用aws-xray-recorder-sdk-aws-sdk，自动对所有 适用于 Java 的 AWS SDK 客户进行仪器。
- aws-xray-recorder-sdk-aws-sdk-v2-instrumentor— 使用aws-xray-recorder-sdk-aws-sdk-v2，自动检测所有 适用于 Java 的 AWS SDK 2.2 及更高版本的客户端。
- aws-xray-recorder-sdk-apache-http - 检测使用 Apache HTTP 客户端进行的传出 HTTP 调用。
- aws-xray-recorder-sdk-spring - 为 Spring AOP 框架应用程序提供拦截程序。
- aws-xray-recorder-sdk-sql-postgres - 检测由 JDBC 对 PostgreSQL 数据库进行的传出调用。
- aws-xray-recorder-sdk-sql-mysql - 检测由 JDBC 对 MySQL 数据库进行的传出调用。
- aws-xray-recorder-sdk-bom - 提供材料清单，您可以用它来指定用于所有子模块的版本。
- aws-xray-recorder-sdk-metrics— 从您收集的 X-Ray 细分中发布未抽样的亚马逊 CloudWatch 指标。

如果您使用 Maven 或 Gradle 来构建应用程序，可将 X-Ray SDK for Java 添加到您的构建配置中。

有关 SDK 的类和方法的参考文档，请参阅AWS X-Ray SDK for Java API 参考。

要求

适用于 Java 的 X-Ray SDK 需要 Java 8 或更高版本、Servlet API 3、 AWS SDK 和 Jackson。

该 SDK 在编译和运行时依赖于以下库：

- AWS 适用于 Java 的 SDK 版本 1.11.398 或更高版本

- Servlet API 3.1.0

这些依赖项在 SDK 的 `pom.xml` 文件中声明，如果您使用 Maven 或 Gradle 生成则自动包括在内。

如果您使用包括在 X-Ray SDK for Java 中的库，则必须使用包括的版本。例如，如果您在运行时已经依赖于 Jackson 并在部署中为该依赖项包括了 JAR 文件，则必须删除这些 JAR 文件，因为 SDK JAR 包括其自己的 Jackson 库版本。

依赖关系管理

可从 Maven 获得 X-Ray SDK for Java：

- 组 – `com.amazonaws`
- 构件 – `aws-xray-recorder-sdk-bom`
- 版本：`2.11.0`

如果您使用 Maven 来生成应用程序，则在 `pom.xml` 文件中添加 SDK 作为依赖项。

Example pom.xml - 依赖项

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>aws-xray-recorder-sdk-bom</artifactId>
      <version>2.11.0</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
<dependencies>
  <dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-xray-recorder-sdk-core</artifactId>
  </dependency>
  <dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-xray-recorder-sdk-apache-http</artifactId>
  </dependency>
  <dependency>
```

```
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-xray-recorder-sdk-aws-sdk</artifactId>
  </dependency>
  <dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-xray-recorder-sdk-aws-sdk-instrumentor</artifactId>
  </dependency>
  <dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-xray-recorder-sdk-sql-postgres</artifactId>
  </dependency>
  <dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-xray-recorder-sdk-sql-mysql</artifactId>
  </dependency>
</dependencies>
```

对于 Gradle，添加 SDK 作为 `build.gradle` 文件中的编译时依赖项。

Example build.gradle - 依赖项

```
dependencies {
  compile("org.springframework.boot:spring-boot-starter-web")
  testCompile("org.springframework.boot:spring-boot-starter-test")
  compile("com.amazonaws:aws-java-sdk-dynamodb")
  compile("com.amazonaws:aws-xray-recorder-sdk-core")
  compile("com.amazonaws:aws-xray-recorder-sdk-aws-sdk")
  compile("com.amazonaws:aws-xray-recorder-sdk-aws-sdk-instrumentor")
  compile("com.amazonaws:aws-xray-recorder-sdk-apache-http")
  compile("com.amazonaws:aws-xray-recorder-sdk-sql-postgres")
  compile("com.amazonaws:aws-xray-recorder-sdk-sql-mysql")
  testCompile("junit:junit:4.11")
}
dependencyManagement {
    imports {
        mavenBom('com.amazonaws:aws-java-sdk-bom:1.11.39')
        mavenBom('com.amazonaws:aws-xray-recorder-sdk-bom:2.11.0')
    }
}
```

如果您使用 Elastic Beanstalk 来部署应用程序，则可以使用 Maven 或 Gradle 在每次部署时生成 on-instance，而不是生成和上传包括所有依赖项的大档案。有关使用 Gradle 的示例，请参阅示例应用程序。

适用于 Java 的AWS X-Ray 自动检测代理

适用于 Java 的 AWS X-Ray 自动检测代理是一种跟踪解决方案，只需最少的开发工作即可对 Java Web 应用程序进行检测。该代理能够跟踪基于 Servlet 的应用程序，以及使用支持的框架和库发出的该代理所有的下游请求。其中包括下游 Apache HTTP 请求、 AWS SDK 请求以及使用 JDBC 驱动程序进行的 SQL 查询。该代理跨线程传播 X-Ray 上下文，包括所有活动分段和子分段。Java 代理仍然可以使用 X-Ray SDK 的所有配置和多功能性。选择的都是合适的默认值以确保轻松就可以使用该代理。

X-Ray 代理解决方案最适合基于 Servlet、请求响应 Java Web 应用程序服务器。如果您的应用程序使用异步框架或者不能很好地建模为请求-响应服务，则可能需要考虑改用 SDK 进行手动检测。

X-Ray 代理是使用分布式系统理解工具包（简称 DiSCo）构建的。Di SCo 是一个开源框架，用于构建可在分布式系统中使用的 Java 代理。虽然使用 X-Ray 代理不需要了解 DiSCo ，但您可以通过访问其主页来了解有关该项目的更多信息 GitHub。X-Ray 代理也是完全开源的。要查看源代码、做出贡献或提出有关代理的问题，请访问代理的存储库 GitHub。

示例应用程序

该eb-java-scorekeep示例应用程序适用于使用 X-Ray 代理进行检测。此分支不包含 Servlet 筛选器或记录器配置，因为这些功能由代理完成。若要在本地运行该应用程序或使用 AWS 资源，请按照示例应用程序的自述文件中列出的步骤操作。关于如何使用示例应用程序生成 X-Ray 跟踪的说明位于示例应用程序的教程中。

入门

请按照以下步骤操作，开始在您自己的应用程序中使用 X-Ray 自动检测 Java 代理。

1. 在环境中运行 X-Ray 进程守护程序。有关更多信息，请参阅 AWS X-Ray 守护程序。
2. 下载代理的最新发行版。解压缩存档并记下其在文件系统中的位置。其内容应与以下内容类似。

```
disco
### disco-java-agent.jar
### disco-plugins
    ### aws-xray-agent-plugin.jar
    ### disco-java-agent-aws-plugin.jar
    ### disco-java-agent-sql-plugin.jar
    ### disco-java-agent-web-plugin.jar
```

3. 修改应用程序的 JVM 参数使其包含以下内容，以启用代理。如适用，请确保将 `-javaagent` 参数放在 `-jar` 参数之前。修改 JVM 参数的过程因启动 Java 服务器所使用的工具和框架而异。请参阅服务器框架的文档了解详细指南。

```
-javaagent:/<path-to-disco>/disco-java-agent.jar=pluginPath=/<path-to-disco>/disco-plugins
```

4. 设置 AWS_XRAY_TRACING_NAME 环境变量或 com.amazonaws.xray.strategy.tracingName 系统属性以指定您的应用程序在 X-Ray 控制台上的显示方式。如果未提供名称，则使用默认名称。
5. 重启服务器或容器。现在，会跟踪传入的请求及其下游调用。如果没有看到预期结果，请参阅the section called “故障排除”。

配置

X-Ray 代理由用户提供的外部 JSON 文件进行配置。默认情况下，此文件位于名为“xray-agent.json”的用户的类路径的根目录中（例如，在其 resources 目录中）。可以将 com.amazonaws.xray.configFile 系统属性设置为配置文件的绝对系统文件路径，为配置文件配置自定义位置。

示例配置文件如下所示。

```
{
    "serviceName": "XRayInstrumentedService",
    "contextMissingStrategy": "LOG_ERROR",
    "daemonAddress": "127.0.0.1:2000",
    "tracingEnabled": true,
    "samplingStrategy": "CENTRAL",
    "traceIdInjectionPrefix": "prefix",
    "samplingRulesManifest": "/path/to/manifest",
    "awsServiceHandlerManifest": "/path/to/manifest",
    "awsSdkVersion": 2,
    "maxStackTraceLength": 50,
    "streamingThreshold": 100,
    "traceIdInjection": true,
    "pluginsEnabled": true,
    "collectSqlQueries": false
}
```

配置规范

下表介绍了每个属性的有效值。属性名称区分大小写，但它们的键不区分大小写。可以被环境变量和系统属性覆盖的属性，其优先级顺序始终先是环境变量、系统属性，然后再是配置文件。有关您可以覆盖的属性的相关信息，请参阅环境变量。所有字段都是可选字段。

属性名称	类型	有效值	描述	环境变量	系统属性	默认
serviceName	字符串	任何字符串	将在 X-Ray 控制台中显示的已检测服务的名称。	AWS_XRAY_TRACING_NAME	com.amazonaws.xray.strategy.tracingNam	XRayInstrumentedService
contextMissingStrategy	字符串	LOG_ERROR、IGNORE_ERROR	代理尝试使用 X-Ray 分段上下文但不存在时所采取的操作。	AWS_XRAY_上下文_缺失	com.amazonaws.xray.strategy.contextMissingStrategy	LOG_ERROR
daemonAddress	字符串	格式化的 IP 地址和端口，或者 TCP 和 UDP 地址列表	代理用于与 X-Ray 进程守护程序通信的地址。	AWS_XRAY_守护进程_地址	com.amazonaws.xray.emitter.daemonAddress	127.0.0.1:2000
tracingEnabled	布尔值	True, False	启用 X-Ray 代理进行检测。	AWS_XRAY_追踪_已启用	com.amazonaws.xray.tracingEnabled	TRUE
samplingStrategy	字符串	CENTRAL、LOCAL、NONE、ALL	代理使用的采样策略。ALL 捕获所有请求，NONE 不捕获任何请求。请参阅采样规则。	不适用	不适用	CENTRAL

属性名称	类型	有效值	描述	环境变量	系统属性	默认
traceIdInjection前缀	字符串	任何字符串	在日志中注入跟踪 IDs 之前包含提供的前缀。	不适用	不适用	无（空字符串）
samplingRulesManifest	字符串	绝对文件路径	自定义采样规则文件的路径，该文件用作本地采样策略的采样规则或中心策略的后备规则的来源。	不适用	不适用	DefaultSamplingRules.json
awsServiceHandler清单	字符串	绝对文件路径	自定义参数允许列表的路径，可从 AWS SDK 客户端捕获其他信息。	不适用	不适用	DefaultOperationParameterWhitelist.json
awsSdkVersion	整数	1、2	您正在使用的 AWS SDK for Java 的版本。如果 `awsServiceHandlerManifest` 也没有设置，请会被忽略。	不适用	不适用	2

属性名称	类型	有效值	描述	环境变量	系统属性	默认
maxStackTrace长度	整数	非负整数	一个跟踪中记录的堆栈跟踪的最大行数。	不适用	不适用	50
streamingThreshold	整数	非负整数	至少在关闭这么多子分段之后，它们会被流式传输到守护程序 out-of-band，以避免区块太大。	不适用	不适用	100
traceIdInjection	布尔值	True, False	如果还添加了日志记录配置中所述的依赖项和配置，则启用 X-Ray 跟踪 ID 注入日志。否则，不执行任何操作。	不适用	不适用	TRUE
pluginsEnabled	布尔值	True, False	启用用于记录有关您正在操作的 AWS 环境的元数据的插件。请参阅插件。	不适用	不适用	TRUE

属性名称	类型	有效值	描述	环境变量	系统属性	默认
collectSqlQueries	布尔值	True, False	尽量将 SQL 查询字符串记录在 SQL 子分段中。	不适用	不适用	FALSE
contextPropagation	布尔值	True, False	如果是 true，则在线程之间自动传播 X-Ray 上下文。否则，需要使用 Thread Local 来存储上下文，并且需要手动跨线程传播。	不适用	不适用	TRUE

日志记录配置

可以按照与 X-Ray SDK for Java 相同的方式配置 X-Ray 代理的日志级别。请参阅 日志记录，详细了解如何使用 X-Ray SDK for Java 配置日志记录。

手动检测

如果希望在代理的自动检测之外再执行手动检测，请将 X-Ray SDK 作为依赖项添加到您的项目。请注意，跟踪传入请求中提及的 SDK 自定义 Servlet 筛选器与 X-Ray 代理不兼容。

> **Note**
>
> 您必须使用最新版本的 X-Ray SDK 来执行手动检测，同时还要使用代理。

如果您正在处理的是 Maven 项目，请将以下依赖项添加到您的 `pom.xml` 文件中。

```
<dependencies>
  <dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-xray-recorder-sdk-core</artifactId>
    <version>2.11.0</version>
  </dependency>
  </dependencies>
```

如果您正在处理的是 Gradle 项目，请将以下依赖项添加到您的 `build.gradle` 文件中。

```
implementation 'com.amazonaws:aws-xray-recorder-sdk-core:2.11.0'
```

在使用代理 IDs时，除了注释、元数据和用户之外，您还可以添加自定义子细分，就像使用普通 SDK 一样。代理会自动跨线程传播上下文，因此在使用多线程应用程序时，无需使用任何解决方法即可传播上下文。

故障排除

由于代理提供全自动检测功能，因此在遇到问题时可能很难确定问题的根本原因。如果 X-Ray 代理无法按预期运行，请查看以下问题和解决方案。X-Ray 代理和 SDK 使用 Jakarta Commons Logging（JCL）。若要查看日志记录输出，请确保将 JCL 桥接到您在类路径上的日志记录后端，如以下示例所示：`log4j-jcl` 或 `jcl-over-slf4j`。

问题：我的应用程序上已经启用了 Java 代理，但在 X-Ray 控制台上却什么都看不到。

X-Ray 进程守护程序是否在同一台计算机上运行？

如果不是，请参阅 X-Ray 进程守护程序文档进行设置。

在应用程序日志中，是否看到类似于“正在初始化 X-Ray 代理记录器”这样的消息？

如果您已正确将代理添加到应用程序中，则在应用程序启动时，在开始接受请求之前，将在信息级别记录此消息。如果没有出现此消息，则说明您的 Java 进程未运行 Java 代理。确保您已正确执行所有设置步骤，不存在错别字。

在您的应用程序日志中，您是否看到几条错误消息，上面写着 “禁止 AWS X-Ray 上下文缺失异常” 之类的内容？

之所以出现这些错误，是因为代理正在尝试检测下游请求，例如 AWS SDK 请求或 SQL 查询，但代理无法自动创建区段。如果您看到其中许多错误消息，则代理可能并不是最适合您用例的工具，可能需要

考虑改用 X-Ray SDK 进行手动检测。或者，可以启用 X-Ray SDK 调试日志，以查看上下文缺失异常发生位置的堆栈跟踪。可以用自定义分段封装代码的这些部分，这样可以解决这些错误。请参阅检测启动代码中的示例代码，查看使用自定义分段包装下游请求的示例。

问题：我预期的一些分段没有出现在 X-Ray 控制台上

您的应用程序是否使用多线程？

如果您希望创建的某些分段未出现在控制台上，则可能是应用程序中的后台线程造成的。如果您的应用程序使用 “触发和忘记” 的后台线程执行任务，例如使用软件开发工具包一次性调用 Lambda 函数，或者定期轮询某些 HTTP 端点，则在代理跨线程传播上下文时，可能会使代理感到困惑。 AWS 若要验证您遇到的是不是这个问题，可以启用 X-Ray SDK 调试日志并查看如下所示的消息：未发射名为 <NAME> 的分段，因为它是正在进行中的子分段的父级。若要解决此问题，可以尝试在服务器返回前加入后端线程以确保记录下在其中完成的全部工作。或者，也可以将代理的 `contextPropagation` 配置设置为 `false`，在后台禁用上下文传播。如果这样做，则必须使用自定义分段手动检测这些线程，或忽略它们造成的上下文缺失异常。

您是否制定采样规则？

如果 X-Ray 主机上出现了看似随机或意想不到的分段，或者您期望出现在控制台上的分段没有出现，那么可能遇到了采样问题。X-Ray 代理使用 X-Ray 控制台中的规则将集中采样应用于其创建的所有分段。默认规则为每秒 1 个分段，之后再加上 5% 的分段进行采样。这意味着可能不会对使用代理快速创建的区段进行采样。要解决这个问题，应该在 X-Ray 控制台上创建自定义采样规则，对所需的分段进行适当采样。有关更多信息，请参阅采样。

配置 X-Ray SDK for Java

X-Ray SDK for Java 包括提供全局记录器的、名为 `AWSXRay` 的类。这是可用于检测代码的 `TracingHandler`。您可以配置全局记录器以自定义为传入 HTTP 调用创建分段的 `AWSXRayServletFilter`。

Sections

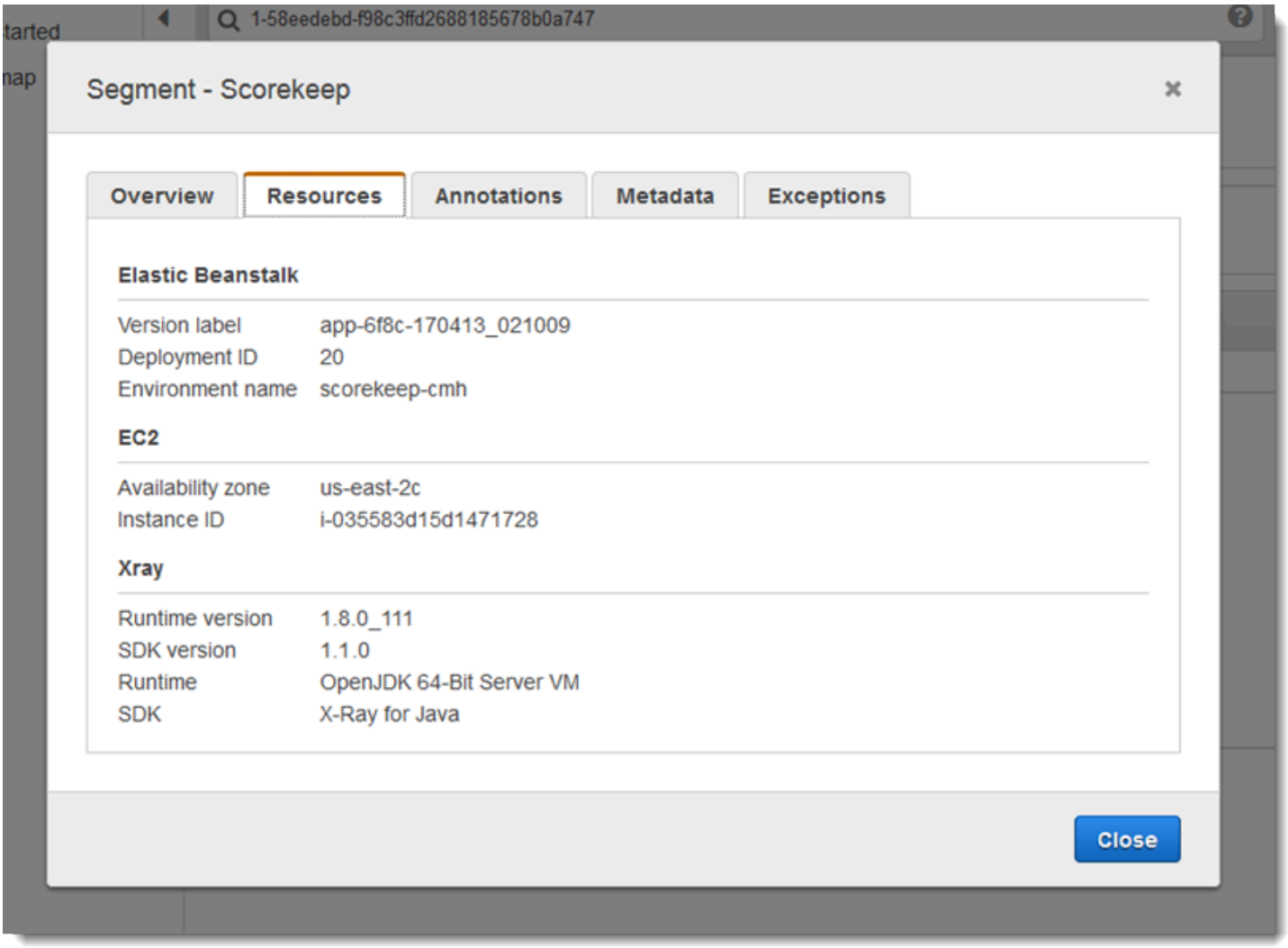
- 服务插件
- 采样规则
- 日志记录
- 分段侦听器
- 环境变量
- 系统属性

服务插件

`plugins` 用于记录有关托管应用程序的服务的信息。

插件

- Amazon EC2 — `EC2Plugin` 添加实例 ID、可用区和 CloudWatch 日志组。
- Elastic Beanstalk - `ElasticBeanstalkPlugin` 添加环境名称、版本标签和部署 ID。
- Amazon ECS — `ECSPlugin` 添加容器 ID。
- Amazon EKS — `EKSPlugin` 添加容器 ID、集群名称、容器 ID 和 CloudWatch 日志组。



要使用插件，请在 `AWSXRayRecorderBuilder` 上调用 `withPlugin`。

Example src/main/java/scorekeep/WebConfig.java-录音机

```
import com.amazonaws.xray.AWSXRay;
import com.amazonaws.xray.AWSXRayRecorderBuilder;
import com.amazonaws.xray.plugins.EC2Plugin;
import com.amazonaws.xray.plugins.ElasticBeanstalkPlugin;
import com.amazonaws.xray.strategy.sampling.LocalizedSamplingStrategy;

@Configuration
public class WebConfig {
...
  static {
    AWSXRayRecorderBuilder builder = AWSXRayRecorderBuilder.standard().withPlugin(new
 EC2Plugin()).withPlugin(new ElasticBeanstalkPlugin());

    URL ruleFile = WebConfig.class.getResource("/sampling-rules.json");
    builder.withSamplingStrategy(new LocalizedSamplingStrategy(ruleFile));

    AWSXRay.setGlobalRecorder(builder.build());
  }
}
```

该 SDK 还使用插件设置为设置分段上的 `origin` 字段。这表示运行应用程序的 AWS 资源类型。当您使用多个插件时，SDK 使用以下解析顺序来确定来源： ElasticBeanstalk > EKS > ECS > EC2。

采样规则

该 SDK 使用您在 X-Ray 控制台中定义的采样规则来确定要记录的请求。默认规则跟踪每秒的第一个请求，以及所有将跟踪发送到 X-Ray 的服务的任何其他请求的百分之五。[在 X-Ray 控制台中创建其他规则](#)以自定义为每个应用程序记录的数据量。

该 SDK 按照定义的顺序应用自定义规则。如果请求与多个自定义规则匹配，则 SDK 仅应用第一条规则。

> **Note**
>
> 如果 SDK 无法访问 X-Ray 来获取采样规则，它将恢复为默认的本地规则，即每秒第一个请求以及每个主机所有其他请求的百分之五。如果主机无权调用采样，或者无法连接到 X-Ray 守护程序 APIs，后者充当 SDK 发出的 API 调用的 TCP 代理，则可能会发生这种情况。

您还可以将 SDK 配置为从 JSON 文档加载采样规则。在 X-Ray 采样不可用的情况下，SDK 可以使用本地规则作为备份，也可以只使用本地规则。

Example sampling-rules.json

```
{
  "version": 2,
  "rules": [
    {
      "description": "Player moves.",
      "host": "*",
      "http_method": "*",
      "url_path": "/api/move/*",
      "fixed_target": 0,
      "rate": 0.05
    }
  ],
  "default": {
    "fixed_target": 1,
    "rate": 0.1
  }
}
```

此示例定义了一个自定义规则和一个默认规则。自定义规则采用百分之五的采样率，对于 `/api/move/` 之下的路径要跟踪的请求数量不设下限。默认规则中每秒的第一个请求以及其他请求的百分之十。

在本地定义规则的缺点是，固定目标由记录器的每个实例独立应用而不是由 X-Ray 服务管理。随着您部署更多主机，固定速率会成倍增加，这使得控制记录的数据量变得更加困难。

开启后 AWS Lambda，您无法修改采样率。如果您的函数由检测服务调用，Lambda 将记录生成由该服务采样的请求的调用。如果启用了活动跟踪且不存在任何跟踪标头，则 Lambda 会做出采样决定。

要在 Spring 中提供备份规则，请使用配置类中的 `CentralizedSamplingStrategy` 配置全局记录器。

Example src/main/java/myapp/WebConfig.java-录制器配置

```
import com.amazonaws.xray.AWSXRay;
import com.amazonaws.xray.AWSXRayRecorderBuilder;
import com.amazonaws.xray.javax.servlet.AWSXRayServletFilter;
import com.amazonaws.xray.plugins.EC2Plugin;
```

```
import com.amazonaws.xray.strategy.sampling.LocalizedSamplingStrategy;

@Configuration
public class WebConfig {

  static {
  AWSXRayRecorderBuilder builder = AWSXRayRecorderBuilder.standard().withPlugin(new
 EC2Plugin());

  URL ruleFile = WebConfig.class.getResource("/sampling-rules.json");
  builder.withSamplingStrategy(new CentralizedSamplingStrategy(ruleFile));

  AWSXRay.setGlobalRecorder(builder.build());
}
```

对于 Tomcat，添加一个扩展 `ServletContextListener` 的侦听器，并在部署描述符中注册该侦听器。

Example src/com/myapp/web/Startup.java

```
import com.amazonaws.xray.AWSXRay;
import com.amazonaws.xray.AWSXRayRecorderBuilder;
import com.amazonaws.xray.plugins.EC2Plugin;
import com.amazonaws.xray.strategy.sampling.LocalizedSamplingStrategy;

import java.net.URL;
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;

public class Startup implements ServletContextListener {

    @Override
    public void contextInitialized(ServletContextEvent event) {
        AWSXRayRecorderBuilder builder =
 AWSXRayRecorderBuilder.standard().withPlugin(new EC2Plugin());

        URL ruleFile = Startup.class.getResource("/sampling-rules.json");
        builder.withSamplingStrategy(new CentralizedSamplingStrategy(ruleFile));

        AWSXRay.setGlobalRecorder(builder.build());
    }

    @Override
```

```
    public void contextDestroyed(ServletContextEvent event) { }
}
```

Example WEB-INF/web.xml

```
...
  <listener>
    <listener-class>com.myapp.web.Startup</listener-class>
  </listener>
```

若要仅使用本地规则，请将 `CentralizedSamplingStrategy` 替换为 `LocalizedSamplingStrategy`。

```
builder.withSamplingStrategy(new LocalizedSamplingStrategy(ruleFile));
```

日志记录

默认情况下，SDK 会将 ERROR 级消息输出到应用程序日志。可以在 SDK 上启用调试级别日志记录，将更详细的日志输出到应用程序日志文件。有效的日志级别为 DEBUG、INFO、WARN、ERROR 和 FATAL。FATAL 日志级别会静默所有日志消息，因为 SDK 不会在严重级别记录日志。

Example application.properties

使用 `logging.level.com.amazonaws.xray` 属性设置日志记录级别。

```
logging.level.com.amazonaws.xray = DEBUG
```

当您手动生成子分段时，使用调试日志来识别诸如未结束子分段之类的问题。

跟踪 ID 注入到日志

要将当前完全限定的跟踪 ID 公开到日志语句，您可以将此 ID 注入到映射的诊断上下文 (MDC)。在分段生命周期事件过程中使用 `SegmentListener` 接口从 X-Ray 记录器调用方法。当分段或子分段开始时，使用密钥 `AWS-XRAY-TRACE-ID` 将限定的跟踪 ID 注入到 MDC 中。当该分段结束后，从 MDC 中删除密钥。这会向正在使用的日志库公开跟踪 ID。当子分段结束时，其父级 ID 将注入到 MDC 中。

Example 完全限定的跟踪 ID

完全限定的 ID 表示为 `TraceID@EntityID`

```
1-5df42873-011e96598b447dfca814c156@541b3365be3dafc3
```

此功能适用于使用适用于 Java 的 AWS X-Ray SDK 进行检测的 Java 应用程序，并支持以下日志配置：

- SLF4带有 Logback 后端的 J 前端 API
- SLF4带有 Log4J2 后端的 J 前端 API
- 带有 Log4J2 后端的 Log4J2 前端 API

请查看以下选项卡，了解每个前端和每个后端的需求。

SLF4J Frontend

1. 将以下 Maven 依赖项添加到您的项目中。

```
<dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-xray-recorder-sdk-slf4j</artifactId>
    <version>2.11.0</version>
</dependency>
```

2. 构建 `AWSXRayRecorder` 时包含 `withSegmentListener` 方法。这会添加一个`SegmentListener`类，该类会自动向 SLF4 J MDC 注 IDs 入新的轨迹。

   `SegmentListener` 采用可选字符串作为参数来配置日志语句前缀。可以通过以下方式配置前缀：

   - 无 - 使用默认 `AWS-XRAY-TRACE-ID` 前缀。
   - 空 使用空字符串（例如 `""`）。
   - 自定义 - 使用在字符串中定义的自定义前缀。

   Example **`AWSXRayRecorderBuilder`** statement

```
AWSXRayRecorderBuilder builder = AWSXRayRecorderBuilder
        .standard().withSegmentListener(new SLF4JSegmentListener("CUSTOM-
PREFIX"));
```

Log4J2 front end

1. 将以下 Maven 依赖项添加到您的项目中。

```
<dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-xray-recorder-sdk-log4j</artifactId>
    <version>2.11.0</version>
</dependency>
```

2. 构建 AWSXRayRecorder 时包含 withSegmentListener 方法。这将添加一个SegmentListener类，该类会自动向 SLF4 J MDC 注 IDs 入新的完全限定轨迹。

   SegmentListener 采用可选字符串作为参数来配置日志语句前缀。可以通过以下方式配置前缀：

   - 无 - 使用默认 AWS-XRAY-TRACE-ID 前缀。
   - 空 - 使用空字符串（例如 "" ）并删除前缀。
   - 自定义 - 使用在字符串中定义的自定义前缀。

   Example **AWSXRayRecorderBuilder** statement

```
AWSXRayRecorderBuilder builder = AWSXRayRecorderBuilder
        .standard().withSegmentListener(new Log4JSegmentListener("CUSTOM-
PREFIX"));
```

Logback backend

要将跟踪 ID 插入到日志事件中，您必须修改记录器的 PatternLayout，它设置每个日志记录语句的格式。

1. 查找在哪里配置的 patternLayout。您可以通过编程方式或通过 XML 配置文件执行此操作。要了解更多信息，请参阅 Logback 配置。
2. 在 patternLayout 中的任意位置插入 %X{}，将跟踪 ID 插入到未来的日志记录语句中。%X{AWS-XRAY-TRACE-ID} 指示您正在检索的值包含由 MDC 提供的密钥。要 PatternLayouts 在 Logback 中了解更多信息，请参阅PatternLayout。

Log4J2 backend

1. 查找在哪里配置的 patternLayout。您可以通过编程方式执行此操作，也可以通过以 XML、JSON、YAML 或属性格式编写的配置文件来执行此操作。

如需详细了解如何通过配置文件配置 Log4J2，请参阅配置。

如需详细了解如何以编程方式配置 Log4J2，请参阅编程式配置。

2. 在 PatternLayout 中的任意位置插入 %X{}，将跟踪 ID 插入到未来的日志记录语句中。%X{AWS-XRAY-TRACE-ID} 指示您正在检索的值包含由 MDC 提供的密钥。要了解有关 Log4J2 PatternLayouts 的更多信息，请参阅模式布局。

跟踪 ID 注入示例

以下显示了一个经过修改包含跟踪 ID 的 PatternLayout 字符串。跟踪 ID 在线程名称 (%t) 之后和日志级别 (%-5p) 之前输出。

Example **PatternLayout**（带 ID 注入）

```
%d{HH:mm:ss.SSS} [%t] %X{AWS-XRAY-TRACE-ID} %-5p %m%n
```

AWS X-Ray 自动在日志语句中打印密钥和跟踪 ID，便于解析。下面显示了使用已修改的 PatternLayout 的日志语句。

Example 带 ID 注入的日志语句

```
2019-09-10 18:58:30.844 [nio-5000-exec-4]  AWS-XRAY-TRACE-ID:
 1-5d77f256-19f12e4eaa02e3f76c78f46a@1ce7df03252d99e1 WARN 1 - Your logging message
 here
```

日志记录消息本身保存在模式 %m 中，并在调用记录器时设置。

分段侦听器

分段侦听器是一个用于拦截生命周期事件（例如，由 AWSXRayRecorder 生成的分段的开始和结束）的接口。分段侦听器事件函数的实现可能是在使用 onBeginSubsegment 创建所有子分段时向所有子分段添加相同的注释，使用 afterEndSegment 将每个分段发送到进程守护程序后记录一条消息，或者使用 beforeEndSubsegment 记录由 SQL 拦截程序发送的查询，以验证子分段是否代表 SQL 查询，如果是，则添加其他元数据。

要查看 SegmentListener 函数的完整列表，请访问 AWS X-Ray Recorder SDK for Java API 相关文档。

以下示例说明如何在使用 onBeginSubsegment 创建所有子分段时向所有子分段添加一致的注释，以及如何使用 afterEndSegment 在每个分段末尾打印日志消息。

Example MySegmentListener.java

```
import com.amazonaws.xray.entities.Segment;
import com.amazonaws.xray.entities.Subsegment;
import com.amazonaws.xray.listeners.SegmentListener;

public class MySegmentListener implements SegmentListener {
    .....

    @Override
    public void onBeginSubsegment(Subsegment subsegment) {
        subsegment.putAnnotation("annotationKey", "annotationValue");
    }

    @Override
    public void afterEndSegment(Segment segment) {
        // Be mindful not to mutate the segment
        logger.info("Segment with ID " + segment.getId());
    }
}
```

然后，在构建 AWSXRayRecorder 时引用此自定义分段侦听器。

Example AWSXRayRecorderBuilder 声明

```
AWSXRayRecorderBuilder builder = AWSXRayRecorderBuilder
        .standard().withSegmentListener(new MySegmentListener());
```

环境变量

您可以使用环境变量来配置 X-Ray SDK for Java。SDK 支持以下变量。

- AWS_XRAY_CONTEXT_MISSING - 设置为 RUNTIME_ERROR 在您的已检测代码尝试在分段未打开的情况下记录数据时引发异常。

  有效值

  - RUNTIME_ERROR— 引发运行时异常。
  - LOG_ERROR— 记录错误并继续（默认）。

  - IGNORE_ERROR— 忽略错误并继续。

  对于在未打开任何请求时运行的启动代码或者会生成新线程的代码，如果您尝试在其中使用检测过的客户端，则可能发生与缺失分段或子分段相关的错误。

- AWS_XRAY_DAEMON_ADDRESS - 设置 X-Ray 进程守护程序侦听器的主机和端口。默认情况下，SDK 使用用于跟踪数据（UDP）和采样（TCP）的 127.0.0.1:2000。如果您已将进程守护程序配置为侦听不同端口或者进程守护程序在另一台主机上运行，则使用此变量。

  格式

  - 同一个端口 — *address*:*port*
  - 不同的端口 — tcp:*address*:*port* udp:*address*:*port*

- AWS_LOG_GROUP - 将日志组的名称设置为与您的应用程序关联的日志组。如果您的日志组使用与您的应用程序相同的 AWS 账户和区域，X-Ray 将使用此指定的日志组自动搜索应用程序的区段数据。有关日志组的更多信息，请参阅使用日志组和日志流。
- AWS_XRAY_TRACING_NAME - 设置 SDK 用于进行分段的服务名称。覆盖您根据 servlet 筛选器的分段命名策略设置的服务名称。

环境变量覆盖在代码中设置的等效系统属性和值。

系统属性

您可以将系统属性用作环境变量的 JVM 特定替代项。SDK 支持以下属性：

- com.amazonaws.xray.strategy.tracingName - 等效于 AWS_XRAY_TRACING_NAME。
- com.amazonaws.xray.emitters.daemonAddress - 等效于 AWS_XRAY_DAEMON_ADDRESS。
- com.amazonaws.xray.strategy.contextMissingStrategy - 等效于 AWS_XRAY_CONTEXT_MISSING。

如果同时设置系统属性和等效的环境变量，则使用环境变量值。每种方法都会覆盖在代码中设置的值。

使用适用于 Java 的 X-Ray 开发工具包跟踪传入请求

您可以使用 X-Ray SDK 来跟踪您的应用程序在亚马逊或亚马逊 EC2 ECS 中的 EC2 实例上提供的传入 HTTP 请求。 AWS Elastic Beanstalk

使用 `Filter` 检测传入 HTTP 请求。在您添加 X-Ray servlet 筛选器到应用程序时，适用于 Java 的 X-Ray 开发工具包为每个采样请求创建分段。此分段包括 HTTP 请求的计时、方法和处置。其他检测会在此分段上创建子分段。

> **Note**
>
> 对于 AWS Lambda 函数，Lambda 会为每个采样请求创建一个分段。请参阅AWS Lambda 和 AWS X-Ray了解更多信息。

每个分段都有一个名称，用于在服务映射中标识您的应用程序。可以静态命名分段，也可以将 SDK 配置为根据传入请求中的主机标头对其进行动态命名。动态命名允许根据请求中的域名对跟踪进行分组，并且在名称不匹配预期模式时（例如，如果主机标头是伪造的）应用默认名称。

> **转发的请求**
>
> 如果负载均衡器或其他中间将请求转发到您的应用程序，X-Ray 会提取请求 `X-Forwarded-For` 标头中的客户端 IP 而非 IP 数据包中的源 IP。由于转发的请求记录的客户端 IP 可以伪造，因此不应信任。

在转发请求时，SDK 在分段中设置附加字段来指示此行为。如果分段包含设置为 `x_forwarded_for` 的字段 `true`，则从 HTTP 请求的 `X-Forwarded-For` 标头获取客户端 IP。

信息处理程序使用包含以下信息的 `http` 数据块为每个传入请求创建一个分段：

- HTTP 方法 - GET、POST、PUT、DELETE 等。
- 客户端地址 - 发送请求的客户端的 IP 地址。
- 响应代码 - 已完成请求的 HTTP 响应代码。
- 时间 - 开始时间（收到请求时）和结束时间（发送响应时）。
- 用户代理 - 请求中的 `user-agent`。
- 内容长度 - 响应中的 `content-length`。

Sections

- 向应用程序中添加跟踪筛选器 (Tomcat)
- 为您的应用程序添加跟踪筛选器 (Spring)

- 配置分段命名策略

向应用程序中添加跟踪筛选器 (Tomcat)

对于 Tomcat，请将 `<filter>` 添加到您项目的 `web.xml` 文件。使用 `fixedName` 参数可指定要应用于为传入请求创建的分段的服务名称。

Example WEB-INF/web.xml - Tomcat

```
<filter>
  <filter-name>AWSXRayServletFilter</filter-name>
  <filter-class>com.amazonaws.xray.javax.servlet.AWSXRayServletFilter</filter-class>
  <init-param>
    <param-name>fixedName</param-name>
    <param-value>MyApp</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>AWSXRayServletFilter</filter-name>
  <url-pattern>*</url-pattern>
</filter-mapping>
```

为您的应用程序添加跟踪筛选器 (Spring)

对于 Spring，请将 `Filter` 添加到您的 `WebConfig` 类。将分段名称作为字符串传递到 `AWSXRayServletFilter` 构造函数。

Example src/main/java/myapp/WebConfig.java-春季

```
package myapp;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Bean;
import javax.servlet.Filter;
import com.amazonaws.xray.javax.servlet.AWSXRayServletFilter;

@Configuration
public class WebConfig {

  @Bean
  public Filter TracingFilter() {
    return new AWSXRayServletFilter("Scorekeep");
  }
```

```
}
```

配置分段命名策略

AWS X-Ray 使用服务名称来标识您的应用程序，并将其与您的应用程序使用的其他应用程序、数据库 APIs、外部数据库和 AWS 资源区分开来。当 X-Ray SDK 为传入请求生成分段时，会将应用程序的服务名称记录在分段的名称字段中。

X-Ray SDK 可以用在 HTTP 请求标头中的 hostname 来命名分段。不过，此标头可以伪造，会导致服务地图中出现意料之外的节点。为防止 SDK 由于包含伪造的主机标头的请求而错误地命名分段，必须为传入请求指定一个默认名称。

如果应用程序为多个域的请求提供服务，则可以将 SDK 配置为使用动态命名策略以在分段名称中反映出这一点。动态命名策略允许 SDK 将主机名用于符合预期模式的请求，并将默认名称应用于不符合预期模式的请求。

例如，可能有一款应用程序为发送到三个子域的请求提供服务，分别为 `www.example.com`、`api.example.com` 和 `static.example.com`。可以使用格式 `*.example.com` 的动态命名策略以识别包含不同名称的子域的分段，服务地图上因此会显示三个服务节点。如果应用程序收到包含与该格式不匹配的 hostname 的请求，您将会在服务地图上看到第四个节点，以及您指定的回退名称。

要对所有请求分段使用同一名称，可在初始化 servlet 筛选器时指定应用程序名称，如上一部分中所示。这与SegmentNamingStrategy通过调用`SegmentNamingStrategy.fixed()`并传递给`AWSXRayServletFilter`构造函数来创建固定值具有相同的效果。

> **Note**
>
> 您可以使用 `AWS_XRAY_TRACING_NAME` 环境变量覆盖您在代码中定义的默认服务名称。

动态命名策略定义一个主机名应匹配的模式和一个在 HTTP 请求中的主机名与该模式不匹配时要使用的默认名称。要在 Tomcat 中动态命名分段，可使用 `dynamicNamingRecognizedHosts` 和 `dynamicNamingFallbackName` 相应地定义模式和默认名称。

Example WEB-INF/web.xml - 带动态命名的 Servlet 筛选器

```
<filter>
  <filter-name>AWSXRayServletFilter</filter-name>
```

```
  <filter-class>com.amazonaws.xray.javax.servlet.AWSXRayServletFilter</filter-class>
  <init-param>
    <param-name>dynamicNamingRecognizedHosts</param-name>
    <param-value>*.example.com</param-value>
  </init-param>
  <init-param>
    <param-name>dynamicNamingFallbackName</param-name>
    <param-value>MyApp</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>AWSXRayServletFilter</filter-name>
  <url-pattern>*</url-pattern>
</filter-mapping>
```

对于 Spring，SegmentNamingStrategy通过调用创建一个动态SegmentNamingStrategy.dynamic()，然后将其传递给AWSXRayServletFilter构造函数。

Example src/main/java/myapp/WebConfig.java-带动态命名的 servlet 过滤器

```
package myapp;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Bean;
import javax.servlet.Filter;
import com.amazonaws.xray.javax.servlet.AWSXRayServletFilter;
import com.amazonaws.xray.strategy.SegmentNamingStrategy;

@Configuration
public class WebConfig {

  @Bean
  public Filter TracingFilter() {
    return new AWSXRayServletFilter(SegmentNamingStrategy.dynamic("MyApp",
 "*.example.com"));
  }
}
```

使用适用于 Java 的 X-Ray SD AWS K 追踪 SDK 调用

当您的应用调用 AWS 服务 以存储数据、写入队列或发送通知时，适用于 Java 的 X-Ray SDK 会按子分段跟踪下游的调用。所跟踪的 AWS 服务 以及您在这些服务中访问的资源（例如，Amazon S3 存储桶或 Amazon SQS 队列），在 X-Ray 控制台的跟踪地图上显示为下游节点。

当您在生成中包括 aws-sdk 和 aws-sdk-instrumentor 子模块时，X-Ray SDK for Java 自动检测所有 AWS SDK 客户端。如果您未包括 Instrumentor 子模块，则可以选择检测一些客户端，同时排除另一些。

要检测单个客户端，请从版本中移除aws-sdk-instrumentor子模块，然后使用该服务的客户端生成器在 AWS SDK 客户端TracingHandler上添加一个XRayClient。

例如，要检测 AmazonDynamoDB 客户端，请将跟踪处理程序传递到 AmazonDynamoDBClientBuilder。

Example MyModel.java-DynamoDB 客户端

```
import com.amazonaws.xray.AWSXRay;
import com.amazonaws.xray.handlers.TracingHandler;

...
public class MyModel {
  private AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard()
        .withRegion(Regions.fromName(System.getenv("AWS_REGION")))
        .withRequestHandlers(new TracingHandler(AWSXRay.getGlobalRecorder()))
        .build();
...
```

对于所有服务，都可以在 X-Ray 控制台中看到调用的 API 的名称。X-Ray 开发工具包会为一部分服务将信息添加到分段，从而在服务地图中提供更高的粒度。

例如，当使用经过检测的 DynamoDB 客户端发出调用时，对于针对表的调用，开发工具包会将表名称添加到分段中。在控制台中，每个表在服务地图中显示为一个独立的节点，以及没有表作为目标的调用的一般 DynamoDB 节点。

Example 对 DynamoDB 进行调用以保存项目的子分段

```
{
  "id": "24756640c0d0978a",
  "start_time": 1.480305974194E9,
  "end_time": 1.4803059742E9,
  "name": "DynamoDB",
  "namespace": "aws",
  "http": {
    "response": {
      "content_length": 60,
```

```
      "status": 200
    }
  },
  "aws": {
    "table_name": "scorekeep-user",
    "operation": "UpdateItem",
    "request_id": "UBQNSO5AEM8T4FDA4RQDEB94OVTDRVV4K4HIRGVJF66Q9ASUAAJG",
  }
}
```

在您访问指定的资源时，对以下服务的调用会在服务地图中创建额外的节点。没有定向到特定资源的调用，为服务创建了通用节点。

- Amazon DynamoDB - 表名称
- Amazon Simple Storage Service - 存储桶和键名称
- Amazon Simple Queue Service - 队列名称

要 AWS 服务 使用 适用于 Java 的 AWS SDK 2.2 及更高版本检测对的下游调用，可以在编译配置中省略该`aws-xray-recorder-sdk-aws-sdk-v2-instrumentor`模块。改为包含 `aws-xray-recorder-sdk-aws-sdk-v2 module`，然后通过为它们配置 `TracingInterceptor` 来检测各个客户端。

Example 适用于 Java 的 AWS SDK 2.2 及更高版本-追踪拦截器

```
import com.amazonaws.xray.interceptors.TracingInterceptor;
import software.amazon.awssdk.core.client.config.ClientOverrideConfiguration
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
//...
public class MyModel {
private DynamoDbClient client = DynamoDbClient.builder()
.region(Region.US_WEST_2)
.overrideConfiguration(ClientOverrideConfiguration.builder()
.addExecutionInterceptor(new TracingInterceptor())
.build()
)
.build();
//...
```

使用适用于 Java 的 X-Ray 开发工具包跟踪对下游 HTTP Web 服务的调用

当您的应用程序调用微服务或公共 HTTP 时 APIs，您可以使用适用于 Java 的 X-Ray SDK 版本`HttpClient`来检测这些调用，并将该 API 作为下游服务添加到服务图中。

适用于 Java 的 X-Ray SDK 包括`DefaultHttpClient`一些`HttpClientBuilder`类，这些类可以用来代替 Apache HttpComponents 等效项来检测传出的 HTTP 调用。

- `com.amazonaws.xray.proxies.apache.http.DefaultHttpClient` - `org.apache.http.impl.client.DefaultHttpClient`
- `com.amazonaws.xray.proxies.apache.http.HttpClientBuilder` - `org.apache.http.impl.client.HttpClientBuilder`

这些库位于 aws-xray-recorder-sdk-apache-http 子模块中。

您可以使用 X-Ray 等效项替换现有的导入语句来检测所有客户端，或者在您初始化客户端以检测特定客户端时使用完全限定名称。

Example HttpClientBuilder

```
import com.fasterxml.jackson.databind.ObjectMapper;
import org.apache.http.HttpEntity;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.util.EntityUtils;
import com.amazonaws.xray.proxies.apache.http.HttpClientBuilder;
...
  public String randomName() throws IOException {
    CloseableHttpClient httpclient = HttpClientBuilder.create().build();
    HttpGet httpGet = new HttpGet("http://names.example.com/api/");
    CloseableHttpResponse response = httpclient.execute(httpGet);
    try {
      HttpEntity entity = response.getEntity();
      InputStream inputStream = entity.getContent();
      ObjectMapper mapper = new ObjectMapper();
      Map<String, String> jsonMap = mapper.readValue(inputStream, Map.class);
      String name = jsonMap.get("name");
      EntityUtils.consume(entity);
      return name;
    } finally {
```

```
      response.close();
    }
  }
```

在您检测对下游 Web API 的调用时，适用于 Java 的 X-Ray 开发工具包会使用有关 HTTP 请求和响应的信息记录子分段。X-Ray 使用子分段为远程 API 生成推断分段。

Example 下游 HTTP 调用的子分段

```
{
  "id": "004f72be19cddc2a",
  "start_time": 1484786387.131,
  "end_time": 1484786387.501,
  "name": "names.example.com",
  "namespace": "remote",
  "http": {
    "request": {
      "method": "GET",
      "url": "https://names.example.com/"
    },
    "response": {
      "content_length": -1,
      "status": 200
    }
  }
}
```

Example 下游 HTTP 调用的推断分段

```
{
  "id": "168416dc2ea97781",
  "name": "names.example.com",
  "trace_id": "1-62be1272-1b71c4274f39f122afa64eab",
  "start_time": 1484786387.131,
  "end_time": 1484786387.501,
  "parent_id": "004f72be19cddc2a",
  "http": {
    "request": {
      "method": "GET",
      "url": "https://names.example.com/"
    },
    "response": {
      "content_length": -1,
```

```
      "status": 200
    }
  },
  "inferred": true
}
```

使用适用于 Java 的 X-Ray 开发工具包跟踪 SQL 查询

SQL 拦截器

通过将适用于 Java 的 X-Ray 开发工具包 JDBC 拦截程序添加到数据源配置来检测 SQL 数据库查询。

- PostgreSQL – `com.amazonaws.xray.sql.postgres.TracingInterceptor`
- MySQL – `com.amazonaws.xray.sql.mysql.TracingInterceptor`

这些拦截程序分别位于 aws-xray-recorder-sql-postgres 和 aws-xray-recorder-sql-mysql 子模块中。它们实现 `org.apache.tomcat.jdbc.pool.JdbcInterceptor` 并与 Tomcat 连接池兼容。

> **Note**
>
> 为了安全起见，SQL 拦截程序不在子分段中记录 SQL 查询本身。

对于 Spring，在属性文件中添加拦截程序并使用 Spring Boot 的 `DataSourceBuilder` 构建数据源。

Example **`src/main/java/resources/application.properties`** - ostgreSQL JDBC 拦截器

```
spring.datasource.continue-on-error=true
spring.jpa.show-sql=false
spring.jpa.hibernate.ddl-auto=create-drop
spring.datasource.jdbc-interceptors=com.amazonaws.xray.sql.postgres.TracingInterceptor
spring.jpa.database-platform=org.hibernate.dialect.PostgreSQL94Dialect
```

Example **`src/main/java/myapp/WebConfig.java`** - 数据源

```
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import org.springframework.boot.autoconfigure.jdbc.DataSourceBuilder;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.context.annotation.Bean;
```

```
import org.springframework.context.annotation.Configuration;
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;

import javax.servlet.Filter;
import javax.sql.DataSource;
import java.net.URL;

@Configuration
@EnableAutoConfiguration
@EnableJpaRepositories("myapp")
public class RdsWebConfig {

  @Bean
  @ConfigurationProperties(prefix = "spring.datasource")
  public DataSource dataSource() {
      logger.info("Initializing PostgreSQL datasource");
      return DataSourceBuilder.create()
              .driverClassName("org.postgresql.Driver")
              .url("jdbc:postgresql://" + System.getenv("RDS_HOSTNAME") + ":" +
 System.getenv("RDS_PORT") + "/ebdb")
              .username(System.getenv("RDS_USERNAME"))
              .password(System.getenv("RDS_PASSWORD"))
              .build();
  }
...
}
```

对于 Tomcat，使用对适用于 Java 类的 X-Ray 开发工具包的引用来对 JDBC 数据源调用 `setJdbcInterceptors`。

Example **`src/main/myapp/model.java`** - 数据源

```
import org.apache.tomcat.jdbc.pool.DataSource;
...
DataSource source = new DataSource();
source.setUrl(url);
source.setUsername(user);
source.setPassword(password);
source.setDriverClassName("com.mysql.jdbc.Driver");
source.setJdbcInterceptors("com.amazonaws.xray.sql.mysql.TracingInterceptor;");
```

Tomcat JDBC 数据源库包含在适用于 Java 的 X-Ray 开发工具包中，但您可以将其声明为提供的依赖关系来记载您将会使用它。

Example **pom.xml** - JDBC 数据源

```
<dependency>
  <groupId>org.apache.tomcat</groupId>
  <artifactId>tomcat-jdbc</artifactId>
  <version>8.0.36</version>
  <scope>provided</scope>
</dependency>
```

原生 SQL 跟踪装饰器

- 将 aws-xray-recorder-sdk-sql 添加到依赖项。
- 装饰您的数据库数据源、连接或语句。

```
dataSource = TracingDataSource.decorate(dataSource)
connection = TracingConnection.decorate(connection)
statement = TracingStatement.decorateStatement(statement)
preparedStatement = TracingStatement.decoratePreparedStatement(preparedStatement,
 sql)
callableStatement = TracingStatement.decorateCallableStatement(callableStatement,
 sql)
```

使用适用于 Java 的 X-Ray 开发工具包生成自定义子分段

子分段可为跟踪的分段扩展为了给请求提供服务而已完成的工作的详细信息。每次使用已检测的客户端进行调用时，X-Ray SDK 在子分段中记录生成的信息。您可以创建其他子分段来分组其他子分段，来度量某个代码段的性能如何，或是来记录注释和元数据。

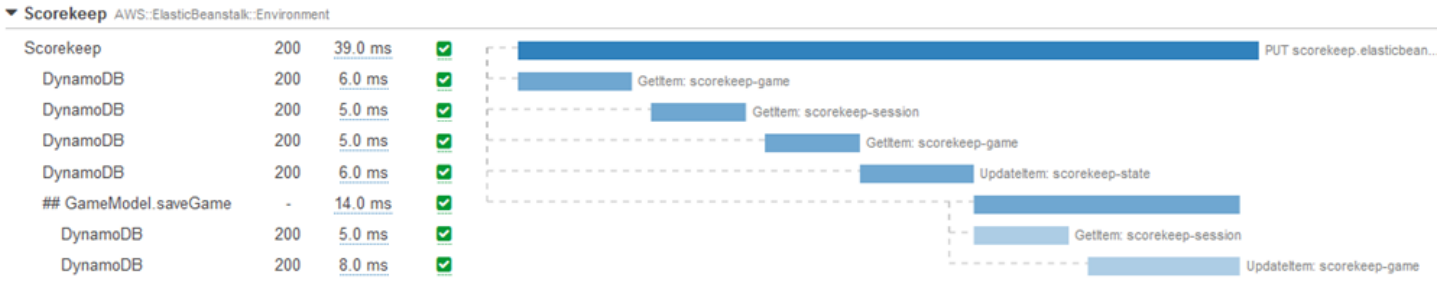
要管理子分段，请使用 beginSubsegment 和 endSubsegment 方法。

Example GameModel.java-自定义子细分

```
import com.amazonaws.xray.AWSXRay;
...
  public void saveGame(Game game) throws SessionNotFoundException {
    // wrap in subsegment
    Subsegment subsegment = AWSXRay.beginSubsegment("Save Game");
    try {
      // check session
      String sessionId = game.getSession();
      if (sessionModel.loadSession(sessionId) == null ) {
```

```
        throw new SessionNotFoundException(sessionId);
      }
      mapper.save(game);
    } catch (Exception e) {
      subsegment.addException(e);
      throw e;
    } finally {
      AWSXRay.endSubsegment();
    }
  }
```

在此示例中，子分段中的代码使用会话模型上的方法从 DynamoDB 加载游戏会话，并使用的 Dynam 适用于 Java 的 AWS SDK oDB 映射器保存游戏。在子分段中包装此代码将调用控制台跟踪视图中 `Save Game` 子分段的 DynamoDB 子项。



如果子分段中的代码引发了检查异常，将其包装在 `try` 代码块中并在 `AWSXRay.endSubsegment()` 代码块中调用 `finally` 以确保始终结束子分段。如果子分段未结束，则父分段无法完成，不发送到 X-Ray。

对于未引发检查异常的代码，可以将代码传递到 `AWSXRay.CreateSubsegment` 作为 lambda 函数。

Example 子分段 Lambda 函数

```
import com.amazonaws.xray.AWSXRay;

AWSXRay.createSubsegment("getMovies", (subsegment) -> {
    // function code
});
```

当您在分段或者其他子分段中创建子分段时，适用于 Java 的 X-Ray 开发工具包将为其生成 ID 并记录开始时间和结束时间。

Example 包含元数据的子分段

```
"subsegments": [{
```

```
  "id": "6f1605cd8a07cb70",
  "start_time": 1.480305974194E9,
  "end_time": 1.4803059742E9,
  "name": "Custom subsegment for UserModel.saveUser function",
  "metadata": {
    "debug": {
      "test": "Metadata string from UserModel.saveUser"
    }
  },
```

对于异步和多线程编程，必须手动将子分段传递给 endSubsegment() 方法以确保其正确关闭，因为在异常执行期间可能会修改 X-Ray 上下文。如果异步子分段在其父分段之前关闭，则此方法将会自动整个分段流式传输到 X-Ray 进程守护程序。

Example 异步子分段

```
@GetMapping("/api")
public ResponseEntity<?> api() {
  CompletableFuture.runAsync(() -> {
      Subsegment subsegment = AWSXRay.beginSubsegment("Async Work");
      try {
          Thread.sleep(3000);
      } catch (InterruptedException e) {
          subsegment.addException(e);
          throw e;
      } finally {
          AWSXRay.endSubsegment(subsegment);
      }
  });
  return ResponseEntity.ok().build();
}
```

使用 X-Ray SDK for Java，将注释和元数据添加到分段

可以利用注释和元数据记录与请求、环境或应用程序相关的其他信息。可以将注释和元数据添加到 X-Ray 开发工具包创建的分段或您创建的自定义子分段。

注释是带字符串、数字或布尔值的键值对。系统会对注释编制索引，以便与筛选表达式一起使用。使用注释记录要用于对控制台中的跟踪进行分组的数据或在调用 GetTraceSummaries API 时使用的数据。

元数据是可以具有任何类型值的键-值对，包括对象和列表，但没有编制索引，无法与筛选条件表达式一起使用。使用元数据记录要存储在跟踪中但不需要用于搜索跟踪的其他数据。

除了注释和元数据之外，您还可以在分段上记录用户 ID 字符串。用户 IDs 被记录在区段的单独字段中，并编制索引以供搜索使用。

Sections

- 使用 X-Ray SDK for Java 记录注释
- 使用 X-Ray SDK for Java 记录元数据
- 使用适用于 Java IDs 的 X-Ray SDK 录制用户

使用 X-Ray SDK for Java 记录注释

使用注释记录有关要为其编制索引以进行搜索的分段和子分段的信息。

注释要求

- 键 - X-Ray 注释的键最多可以包含 500 个字母数字字符。除了点或句点（.）之外，不能使用空格或符号
- 值 - X-Ray 注释的值最多可以包含 1,000 个 Unicode 字符。
- 注释的数量 - 每个跟踪最多可使用 50 条注释。

记录注释

1. 从 AWSXRay 获取对当前分段或子分段的引用。

```
import com.amazonaws.xray.AWSXRay;
import com.amazonaws.xray.entities.Segment;
...
Segment document = AWSXRay.getCurrentSegment();
```

或

```
import com.amazonaws.xray.AWSXRay;
import com.amazonaws.xray.entities.Subsegment;
...
Subsegment document = AWSXRay.getCurrentSubsegment();
```

2. 调用带有字符串键和布尔值、数字值或字符串值的 putAnnotation。

```
document.putAnnotation("mykey", "my value");
```

以下示例说明如何使用包含点和布尔值、数字值或字符串值的字符串键调用 putAnnotation。

```
document.putAnnotation("testkey.test", "my value");
```

开发工具包将注释以键-值对的形式记录在分段文档的 annotations 对象中。使用相同键调用两次 putAnnotation 将覆盖同一分段或子分段上之前记录的值。

要查找具有带特定值的注释的跟踪，请在annotation[*key*]筛选表达式中使用 关键字。

Example **src/main/java/scorekeep/GameModel.java** - 注释和元数据

```
import com.amazonaws.xray.AWSXRay;
import com.amazonaws.xray.entities.Segment;
import com.amazonaws.xray.entities.Subsegment;
...
  public void saveGame(Game game) throws SessionNotFoundException {
    // wrap in subsegment
    Subsegment subsegment = AWSXRay.beginSubsegment("## GameModel.saveGame");
    try {
      // check session
      String sessionId = game.getSession();
      if (sessionModel.loadSession(sessionId) == null ) {
        throw new SessionNotFoundException(sessionId);
      }
      Segment segment = AWSXRay.getCurrentSegment();
      subsegment.putMetadata("resources", "game", game);
      segment.putAnnotation("gameid", game.getId());
      mapper.save(game);
    } catch (Exception e) {
      subsegment.addException(e);
      throw e;
    } finally {
      AWSXRay.endSubsegment();
    }
  }
```

使用 X-Ray SDK for Java 记录元数据

使用元数据记录有关您无需为其编制索引以进行搜索的分段或子分段的信息。元数据值可以是字符串、数字、布尔值或可序列化为 JSON 对象或数组的任何对象。

记录元数据

1. 从 AWSXRay 获取对当前分段或子分段的引用。

```
import com.amazonaws.xray.AWSXRay;
import com.amazonaws.xray.entities.Segment;
...
Segment document = AWSXRay.getCurrentSegment();
```

或

```
import com.amazonaws.xray.AWSXRay;
import com.amazonaws.xray.entities.Subsegment;
...
Subsegment document = AWSXRay.getCurrentSubsegment();
```

2. 调用带有字符串命名空间、字符串键和布尔值、数字值、字符串值或对象值的 putMetadata。

```
document.putMetadata("my namespace", "my key", "my value");
```

或

调用仅带有键和值的 putMetadata。

```
document.putMetadata("my key", "my value");
```

如果您没有指定命名空间，则开发工具包将使用 default。使用相同键调用两次 putMetadata 将覆盖同一分段或子分段上之前记录的值。

Example **src/main/java/scorekeep/GameModel.java** - 注释和元数据

```
import com.amazonaws.xray.AWSXRay;
import com.amazonaws.xray.entities.Segment;
import com.amazonaws.xray.entities.Subsegment;
...
```

```
public void saveGame(Game game) throws SessionNotFoundException {
  // wrap in subsegment
  Subsegment subsegment = AWSXRay.beginSubsegment("## GameModel.saveGame");
  try {
    // check session
    String sessionId = game.getSession();
    if (sessionModel.loadSession(sessionId) == null ) {
      throw new SessionNotFoundException(sessionId);
    }
    Segment segment = AWSXRay.getCurrentSegment();
    subsegment.putMetadata("resources", "game", game);
    segment.putAnnotation("gameid", game.getId());
    mapper.save(game);
  } catch (Exception e) {
    subsegment.addException(e);
    throw e;
  } finally {
    AWSXRay.endSubsegment();
  }
}
```

使用适用于 Java IDs 的 X-Ray SDK 录制用户

记录请求细分中的用户，以识别发送请求的用户。 IDs

要记录用户 IDs

1. 从 AWSXRay 获取对当前分段的引用。

```
import com.amazonaws.xray.AWSXRay;
import com.amazonaws.xray.entities.Segment;
...
Segment document = AWSXRay.getCurrentSegment();
```

2. 使用发送请求的用户的字符串 ID 调用 `setUser`。

```
document.setUser("U12345");
```

您可以在控制器中调用 `setUser` 以便在应用程序开始处理请求后立即记录用户 ID。如果您只打算使用分段来设置用户 ID，可以在单个行中链接这些调用。

Example src/main/java/scorekeep/MoveController.java — 用户 ID

```
import com.amazonaws.xray.AWSXRay;
...
  @RequestMapping(value="/{userId}", method=RequestMethod.POST)
  public Move newMove(@PathVariable String sessionId, @PathVariable String
 gameId, @PathVariable String userId, @RequestBody String move) throws
 SessionNotFoundException, GameNotFoundException, StateNotFoundException,
 RulesException {
    AWSXRay.getCurrentSegment().setUser(userId);
    return moveFactory.newMove(sessionId, gameId, userId, move);
  }
```

要查找用户 ID 的跟踪，请在user筛选表达式中使用 关键字。

AWS X-Ray 适用于 Java 的 X-Ray SDK 的指标

本主题介绍 AWS X-Ray 命名空间、指标和维度。您可以使用适用于 Java 的 X-Ray SDK 从您收集的 X-Ray 细分中发布未采样的亚马逊 CloudWatch 指标。这些指标来自分段的开始和结束时间以及错误、故障和限制状态标志。使用这些指标可暴露子分段里的重试和依赖项问题。

CloudWatch 是一个指标存储库。指标是中的基本概念 CloudWatch ，代表一组按时间排序的数据点。您（或 AWS 服务）将指标数据点发布到其中， CloudWatch 并将有关这些数据点的统计数据作为一组有序的时间序列数据进行检索。

指标通过名称、命名空间以及一个或多个维度进行唯一定义。每个数据点都有一个时间戳和一个可选的度量单位。当请求统计数据时，返回的数据流根据命名空间、指标名称和维度加以识别。

有关的更多信息 CloudWatch，请参阅 Amazon CloudWatch 用户指南。

X-Ray CloudWatch 指标

ServiceMetrics/SDK 命名空间包括以下指标。

指标	可用统计数据	描述	单位
Latency	平均、最小、最大、计数	开始时间和结束时间之间的差异。平均、最小和最大都描述操	毫秒

指标	可用统计数据	描述	单位
		作延迟。计数描述调用次数。	
ErrorRate	平均、总计	导致错误的失败请求率，显示 4xx Client Error 状态码。	百分比
FaultRate	平均、总计	导致故障的失败跟踪率，显示 5xx Server Error 状态码。	百分比
ThrottleRate	平均、总计	返回 429 状态码的受限制跟踪率。这是 ErrorRate 指标的子集。	百分比
OkRate	平均、总计	导致 OK 状态码的被跟踪请求率。	百分比

X 射线 CloudWatch 尺寸

使用下表中的维度来细化所分析的 X-Ray 返回的指标 Java 应用程序。

维度	描述
ServiceType	服务类型，例如 AWS::EC2::Instance 或 NONE（如果为未知）。
ServiceName	服务的规范名称。

启用 X-Ray CloudWatch 指标

使用以下步骤在您的仪器中启用跟踪指标 Java 应用程序的修订。

配置跟踪指标

1. 将aws-xray-recorder-sdk-metrics包添加为 Apache Maven 依赖。有关更多信息，请参阅 X-Ray SDK for Java Submodules。
2. 启用新的 MetricsSegmentListener() 作为全局记录器构建的一部分。

   Example src/com/myapp/web/Startup.java

```
import com.amazonaws.xray.AWSXRay;
import com.amazonaws.xray.AWSXRayRecorderBuilder;
import com.amazonaws.xray.plugins.EC2Plugin;
import com.amazonaws.xray.plugins.ElasticBeanstalkPlugin;
import com.amazonaws.xray.strategy.sampling.LocalizedSamplingStrategy;

@Configuration
public class WebConfig {
...
  static {
    AWSXRayRecorderBuilder builder = AWSXRayRecorderBuilder
                                        .standard()
                                        .withPlugin(new EC2Plugin())
                                        .withPlugin(new ElasticBeanstalkPlugin())
                                        .withSegmentListener(new
 MetricsSegmentListener());

    URL ruleFile = WebConfig.class.getResource("/sampling-rules.json");
    builder.withSamplingStrategy(new LocalizedSamplingStrategy(ruleFile));

    AWSXRay.setGlobalRecorder(builder.build());
  }
}
```

3. 部署 CloudWatch 代理以使用亚马逊弹性计算云（亚马逊 EC2）、亚马逊弹性容器服务（亚马逊 ECS）或亚马逊 Elastic Kubernetes Service（亚马逊 EKS）收集指标：

   - 要配置 Amazon EC2，请参阅安装 CloudWatch 代理。
   - 要配置 Amazon ECS，请参阅使用 Container Insights 监控 Amazon ECS 容器
   - 要配置 Amazon EKS，请参阅使用 Amazon Obs CloudWatch ervability EKS 插件安装 CloudWatch 代理。

4. 将 SDK 配置为与 CloudWatch 代理通信。默认情况下，SDK 通过地址`127.0.0.1`与 CloudWatch 代理进行通信。您可以通过将环境变量或 Java 属性设置为 `address:port` 来配置备用地址。

   Example 环境变量

   ```
   AWS_XRAY_METRICS_DAEMON_ADDRESS=address:port
   ```

   Example Java 属性

   ```
   com.amazonaws.xray.metrics.daemonAddress=address:port
   ```

验证配置

1. 登录 AWS Management Console 并打开 CloudWatch 控制台，网址为https://console.aws.amazon.com/cloudwatch/。
2. 打开指标选项卡以观察指标的情况。
3. （可选）在 CloudWatch 控制台的日志选项卡上，打开`ServiceMetricsSDK`日志组。查找与主机指标相匹配的日志流，然后确认日志消息。

在多线程应用程序中的线程之间传递分段上下文

在您的应用程序中创建新线程时，`AWSXRayRecorder` 不会维护对当前分段或子分段实体的引用。如果您在新话题中使用经过检测的客户端，SDK 会尝试写入不存在的区段，从而导致 SegmentNotFoundException.

为避免在开发过程中抛出异常，您可以为记录器配置一个ContextMissingStrategy提醒它记录错误的。您可以使用代码配置策略 SetContextMissingStrategy，也可以使用环境变量或系统属性配置等效选项。

解决错误的一种方法是使用新分段：在您启动线程时调用 beginSegment，并在您将其关闭时调用 endSegment。如果您正在检测并非为响应 HTTP 请求而运行的代码，例如在您的应用程序启动时运行的代码，这很适合。

如果您使用多个线程来处理传入请求，您可以将当前分段或子分段传递到新线程，并将其提供给全局记录器。这样可以确保对于新线程中记录的信息，相关联的分段与针对该请求记录的其余信息的关联分段

相同。一旦分段在新线程中可用，即可执行任何可使用 `segment.run(() -> { ... })` 方法访问该分段的上下文的可运行。

有关示例，请参阅 在工作线程中使用检测的客户端。

使用 X-Ray 进行异步编程

适用于 Java 的 X-Ray SDK 可以在异步 Java 程序中使用SegmentContextExecutors。SegmentContextExecutor 实现了 Executor 接口，这意味着它可以传递到 a CompletableFuture的所有异步操作中。这样可以确保任何异步操作都将在其上下文中使用正确的分段执行。

Example 示例 App.java：传递 SegmentContextExecutor 给 CompletableFuture

```
DynamoDbAsyncClient client = DynamoDbAsyncClient.create();

AWSXRay.beginSegment();

// ...

client.getItem(request).thenComposeAsync(response -> {
    // If we did not provide the segment context executor, this request would not be
 traced correctly.
    return client.getItem(request2);
}, SegmentContextExecutors.newSegmentContextExecutor());
```

包含 Spring 以及适用于 Java 的 X-Ray 开发工具包的 AOP

本主题介绍如何使用 X-Ray 开发工具包和 Spring Framework 检测应用程序，而不更改其核心逻辑。这意味着现在有一种非侵入性的方法可以检测远程运行的应用程序。 AWS

启用 Spring 中的 AOP

1. 配置 Spring
2. 向您的应用程序添加跟踪筛选器
3. 对代码添加注释或实现接口
4. 激活应用程序中的 X-Ray

配置 Spring

您可以使用 Maven 或 Gradle 将 Spring 配置为使用 AOP 检测您的应用程序。

如果您使用 Maven 来生成应用程序，则在 `pom.xml` 文件中添加以下依赖项。

```
<dependency>
     <groupId>com.amazonaws</groupId>
     <artifactId>aws-xray-recorder-sdk-spring</artifactId>
     <version>2.11.0</version>
</dependency>
```

对于 Gradle，在 `build.gradle` 文件中添加以下依赖项。

```
compile 'com.amazonaws:aws-xray-recorder-sdk-spring:2.11.0'
```

配置 Spring Boot

除了上一节中介绍的 Spring 依赖项，如果您使用的是 Spring Boot，如果尚位在类路径上，请添加以下依赖项。

Maven：

```
<dependency>
     <groupId>org.springframework.boot</groupId>
     <artifactId>spring-boot-starter-aop</artifactId>
     <version>2.5.2</version>
</dependency>
```

Gradle：

```
compile 'org.springframework.boot:spring-boot-starter-aop:2.5.2'
```

向您的应用程序添加跟踪筛选器

将 `Filter` 添加到 `WebConfig` 类。将分段名称作为字符串传递到 `AWSXRayServletFilter` 构造函数。有关跟踪筛选器和检测传入请求的更多信息，请参阅 使用适用于 Java 的 X-Ray 开发工具包跟踪传入请求。

Example src/main/java/myapp/WebConfig.java-春季

```
package myapp;
```

```
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Bean;
import javax.servlet.Filter;
import com.amazonaws.xray.javax.servlet.AWSXRayServletFilter;

@Configuration
public class WebConfig {

  @Bean
  public Filter TracingFilter() {
    return new AWSXRayServletFilter("Scorekeep");
  }
}
```

Jakarta 支持

Spring 6 企业版使用 Jakarta 而非 Java。为支持这一全新命名空间，X-Ray 创建出位于其自己 Jakarta 命名空间里的类的并行集。

对于筛选器类，将 `javax` 替换为 `jakarta`。配置分段命名策略时，如下所示，在命名策略类名称前添加 `jakarta`：

```
package myapp;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Bean;
import jakarta.servlet.Filter;
import com.amazonaws.xray.jakarta.servlet.AWSXRayServletFilter;
import com.amazonaws.xray.strategy.jakarta.SegmentNamingStrategy;

@Configuration
public class WebConfig {
    @Bean
    public Filter TracingFilter() {
        return new AWSXRayServletFilter(SegmentNamingStrategy.dynamic("Scorekeep"));
    }
}
```

对代码添加注释或实现接口

您的类必须使用 `@XRayEnabled` 注释添加注释，或实现 `XRayTraced` 接口。这将告知 AOP 系统包装受影响类的函数以进行 X-Ray 检测。

激活应用程序中的 X-Ray

要激活应用程序中的 X-Ray 跟踪，您的代码必须通过覆盖以下方法来扩展抽象类 `BaseAbstractXRayInterceptor`。

- `generateMetadata` - 此函数允许对附加到当前函数跟踪的元数据进行自定义。默认情况下，执行函数的类名将记录在元数据中。如果您需要其他信息，则可添加更多数据。
- `xrayEnabledClasses` - 此函数为空，并且应保持此状态。它用作告知拦截程序要包装的方法的指示的主机。通过指定使用要跟踪的 `@XRayEnabled` 添加注释的类来定义指示。以下指示语句告知拦截程序包装使用 `@XRayEnabled` 注释添加注释的所有控制器 bean。

```
@Pointcut(“@within(com.amazonaws.xray.spring.aop.XRayEnabled) && bean(*Controller)”)
```

如果项目使用的是 Spring Data JPA，请考虑从 `AbstractXRayInterceptor` 而非 `BaseAbstractXRayInterceptor` 进行扩展。

示例

以下代码扩展抽象类 `BaseAbstractXRayInterceptor`。

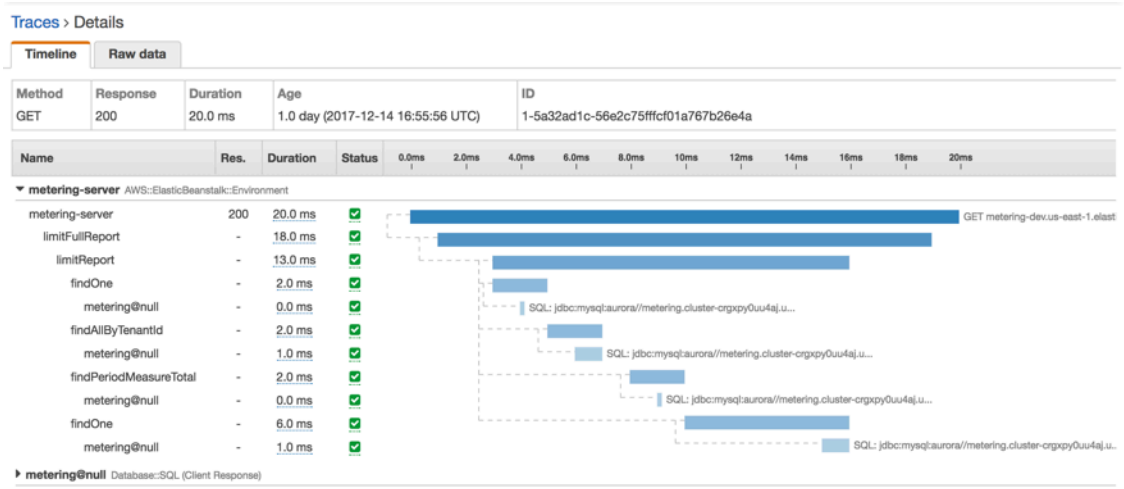
```
@Aspect
@Component
public class XRayInspector extends BaseAbstractXRayInterceptor {
    @Override
    protected Map<String, Map<String, Object>> generateMetadata(ProceedingJoinPoint
 proceedingJoinPoint, Subsegment subsegment) throws Exception {
        return super.generateMetadata(proceedingJoinPoint, subsegment);
    }

  @Override
  @Pointcut("@within(com.amazonaws.xray.spring.aop.XRayEnabled) && bean(*Controller)")

  public void xrayEnabledClasses() {}

}
```

以下代码是一个将由 X-Ray 检测的类。

```
@Service
@XRayEnabled
```

```
public class MyServiceImpl implements MyService {
    private final MyEntityRepository myEntityRepository;

    @Autowired
    public MyServiceImpl(MyEntityRepository myEntityRepository) {
        this.myEntityRepository = myEntityRepository;
    }

    @Transactional(readOnly = true)
    public List<MyEntity> getMyEntities(){
        try(Stream<MyEntity> entityStream = this.myEntityRepository.streamAll()){

            return entityStream.sorted().collect(Collectors.toList());
        }
    }
}
```

如果您已正确配置您的应用程序，则应看到应用程序的完整调用堆栈（从控制器向下至服务调用），如以下控制台屏幕截图所示。



使用 Node.js

有两种方法可用于检测 Node.js 应用程序，以将跟踪数据发送到 X-Ray：

- AWS Distro for OpenTelemetry JavaScript — 提供一组开源库的 AWS 发行版，用于通过 Distro for Collect AWS o r Collector 向多个 AWS 监控解决方案（包括亚马逊和亚马逊 OpenSearch 服务）发送相关的指标和跟踪。 CloudWatch AWS X-Ray OpenTelemetry
- AWS X-Ray Node.js 的 SDK — 一组库，用于通过 X-Ray 守护程序生成跟踪并将其发送到 X-Ray。

有关更多信息，请参阅 在 AWS Distro for 和 X-Ray OpenTelemetry 之间进行选择 SDKs。

AWS 发行版适用于 OpenTelemetry JavaScript

使用 AWS Distro for OpenTelemetry (ADOT) JavaScript，您可以对应用程序进行一次检测，然后将相关的指标和跟踪发送到多个 AWS 监控解决方案 CloudWatch，包括亚马逊 AWS X-Ray和亚马逊服务。 OpenSearch 将 X-Ray 与 AWS Distro 配合使用 OpenTelemetry 需要两个组件：启用 OpenTelemetry SDK 以与 X-Ray 配合使用，以及启用 Collecto OpenTelemetry r 的AWS Distro 以与 X-Ray 配合使用。

要开始使用，请参阅AWS 发行版以获取 OpenTelemetry JavaScript 文档。

> **Note**
>
> 所有服务器端 Node.js 应用程序 JavaScript 都支持 ADOT。AD JavaScript OT 无法将数据从浏览器客户端导出到 X-Ray。

有关将 Distro 用于 with AWS X-Ray 和 other 的 OpenTelemetry 更多信息 AWS 服务，请参阅 AWS Distro for OpenTelemetry 或 AWS Distro for AWS D ocuments。 OpenTelemetry

有关语言支持和用法的更多信息，请参阅上的 O AWS bservability。 GitHub

AWS 适用于 Node.js 的 X-ray SDK

X-Ray SDK for Node.js 是一个面向 Express Web 应用程序和 Node.js Lambda 函数的库，可提供类和方法来生成跟踪数据并将跟踪数据发送给 X-Ray 进程守护程序。跟踪数据包括有关应用程序处理的传入 HTTP 请求的信息，以及应用程序使用 AWS SDK 或 HTTP 客户端对下游服务进行的调用的信息。

> **Note**
>
> X-Ray SDK for Node.js 是一种开源项目，支持 Node.js 14.x 版本及更高版本。你可以关注该项目并在 github 上 GitHub提交议题和拉取请求。 com/aws/aws-xray-sdk-node

如果您使用 Express，首先，在应用程序服务器上添加 SDK 作为中间件来跟踪传入请求。此中间件为每个被跟踪请求创建一个分段并在发送响应时完成该分段。当分段打开时，您可以使用开发工具包客户端的方法将信息添加到分段，并创建子分段以跟踪下游调用。开发工具包还会自动记录在分段打开时应用程序引发的异常。

对于由经过检测的应用程序或服务调用的 Lambda 函数，Lambda 会读取跟踪标头并自动跟踪采样的请求。对于其他函数，您可以将 Lambda 配置为采样和跟踪传入请求。无论哪种情况，Lambda 都会创建分段并将其提供给 X-Ray 开发工具包。

> **Note**
>
> 在 Lambda 上，X-Ray 开发工具包是可选的。如果您不在函数中使用它，您的服务映射仍将包含一个用于 Lambda 服务的节点，以及每个 Lambda 函数的节点。可通过添加该开发工具包检测函数代码，将子分段添加到 Lambda 记录的函数分段。请参阅AWS Lambda 和 AWS X-Ray了解更多信息。

接下来，使用适用于 Node.js 的 X-Ray S AWS DK JavaScript 在 Node.js 客户端中检测你的 SDK。每当您使用已检测的客户端调用下游 AWS 服务 或资源时，SDK 都会在子分段中记录有关该调用的信息。 AWS 服务 您在服务中访问的资源将作为下游节点显示在跟踪地图上，以帮助您识别各个连接上的错误和限制问题。

适用于 Node.js 的 X-Ray SDK 还为对 HTTP Web APIs 和 SQL 查询的下游调用提供了工具。将 HTTP 客户端包含在 SDK 的捕获方法中以记录有关传出 HTTP 调用的信息。对于 SQL 客户端，将捕获方法用于数据库类型。

中间件将采用规则应用于传入请求以确定要跟踪的请求。您可以配置适用于 Node.js 的 X-Ray SDK 来调整采样行为或记录有关运行应用程序的 AWS 计算资源的信息。

记录有关请求以及应用程序在注释和元数据中所做的工作的其他信息。注释是简单的键值对，已为这些键值对编制索引以用于筛选条件表达式，以便您能够搜索包含特定数据的跟踪。元数据条目的限制性较低，并且可以记录整个对象和数组 - 可序列化为 JSON 的任何项目。

> **注释和元数据**
>
> 注释和元数据是您使用 X-Ray 开发工具包添加到分段的任意文本。系统会对注释编制索引，以便与筛选表达式一起使用。元数据未编制索引，但可以使用 X-Ray 控制台或 API 在原始分段中查看。您授予 X-Ray 读取权限的任何人都可以查看这些数据。

当代码中具有大量检测的客户端时，一个请求分段可包含大量子分段，检测的客户端发起的每个调用均对应一个子分段。您可以通过将客户端调用包含在自定义子分段中来整理子分段并为其分组。您可以为整个函数或任何代码部分创建自定义子分段，并记录子分段的元数据和注释，而不是编写父分段的所有内容。

有关 SDK 的类和方法的参考文档，请参阅 AWS X-Ray SDK for Node.js API 参考。

要求

X-Ray SDK for Node.js 需要 Node.js 和以下库：

- `atomic-batcher` - 1.0.2
- `cls-hooked` - 4.2.2
- `pkginfo` - 0.4.0
- `semver` - 5.3.0

在将 SDK 与 NPM 一起安装时，SDK 会拉入这些库。

要跟踪 AWS SDK 客户端，适用于 Node.js 的 X-Ray AWS SDK 需要 Node.js JavaScript 中最低版本的 SDK。

- `aws-sdk` - 2.7.15

依赖关系管理

可从 NPM 获得 X-Ray SDK for Node.js。

- 程序包 - `aws-xray-sdk`

对于本地开发，将 SDK 与 NPM 一起安装在项目目录中。

```
~/nodejs-xray$ npm install aws-xray-sdk
aws-xray-sdk@3.3.3
  ### aws-xray-sdk-core@3.3.3
  # ### @aws-sdk/service-error-classification@3.15.0
  # ### @aws-sdk/types@3.15.0
  # ### @types/cls-hooked@4.3.3
  # # ### @types/node@15.3.0
  # ### atomic-batcher@1.0.2
  # ### cls-hooked@4.2.2
  # # ### async-hook-jl@1.7.6
  # # # ### stack-chain@1.3.7
  # # ### emitter-listener@1.1.2
  # #   ### shimmer@1.2.1
  # ### semver@5.7.1
  ### aws-xray-sdk-express@3.3.3
  ### aws-xray-sdk-mysql@3.3.3
  ### aws-xray-sdk-postgres@3.3.3
```

使用 `--save` 选项可将 SDK 作为应用程序的 `package.json` 中的依赖项保存。

```
~/nodejs-xray$ npm install aws-xray-sdk --save
aws-xray-sdk@3.3.3
```

如果应用程序具有任何其版本与 X-Ray SDK 的依赖项冲突的依赖项，则将会同时安装两个版本以确保兼容性。有关详细信息，请参阅依赖项解析的官方 NPM 文档。

Node.js 示例

使用 AWS X-Ray 适用于 Node.js 的 SDK，在请求通过你的 Node.js 应用程序时 end-to-end查看这些请求。

- Node.js 示例应用程序已启用 GitHub。

配置适用于 Node.js 的 X-Ray 开发工具包

您可以配置带有插件的适用于 Node.js 的 X-Ray 开发工具包 以包括应用程序在其上运行的服务的相关信息，修改默认采样行为，或者添加应用于特定路径请求的采样规则。

Sections

- 服务插件

- 采样规则
- 日志记录
- X-Ray 进程守护程序地址
- 环境变量

服务插件

`plugins` 用于记录有关托管应用程序的服务的信息。

插件

- Amazon EC2 — `EC2Plugin` 添加实例 ID、可用区和 CloudWatch 日志组。
- Elastic Beanstalk - `ElasticBeanstalkPlugin` 添加环境名称、版本标签和部署 ID。
- Amazon ECS — `ECSPlugin` 添加容器 ID。

要使用插件，请使用 `config` 方法配置适用于 Node.js 的 X-Ray 开发工具包客户端。

Example app.js - 插件

```
var AWSXRay = require('aws-xray-sdk');
AWSXRay.config([AWSXRay.plugins.EC2Plugin,AWSXRay.plugins.ElasticBeanstalkPlugin]);
```

该 SDK 还使用插件设置为设置分段上的 `origin` 字段。这表示运行您的应用程序的 AWS 资源类型。当您使用多个插件时，SDK 使用以下解析顺序来确定来源： ElasticBeanstalk > EKS > ECS > EC2。

采样规则

该 SDK 使用您在 X-Ray 控制台中定义的采样规则来确定要记录的请求。默认规则跟踪每秒的第一个请求，以及所有将跟踪发送到 X-Ray 的服务的任何其他请求的百分之五。在 X-Ray 控制台中创建其他规则以自定义为每个应用程序记录的数据量。

该 SDK 按照定义的顺序应用自定义规则。如果请求与多个自定义规则匹配，则 SDK 仅应用第一条规则。

> **Note**
>
> 如果 SDK 无法访问 X-Ray 来获取采样规则，它将恢复为默认的本地规则，即每秒第一个请求以及每个主机所有其他请求的百分之五。如果主机无权调用采样，或者无法连接到 X-Ray 守护程序 APIs，后者充当 SDK 发出的 API 调用的 TCP 代理，则可能会发生这种情况。

您还可以将 SDK 配置为从 JSON 文档加载采样规则。在 X-Ray 采样不可用的情况下，SDK 可以使用本地规则作为备份，也可以只使用本地规则。

Example sampling-rules.json

```
{
  "version": 2,
  "rules": [
    {
      "description": "Player moves.",
      "host": "*",
      "http_method": "*",
      "url_path": "/api/move/*",
      "fixed_target": 0,
      "rate": 0.05
    }
  ],
  "default": {
    "fixed_target": 1,
    "rate": 0.1
  }
}
```

此示例定义了一个自定义规则和一个默认规则。自定义规则采用百分之五的采样率，对于 `/api/move/` 之下的路径要跟踪的请求数量不设下限。默认规则中每秒的第一个请求以及其他请求的百分之十。

在本地定义规则的缺点是，固定目标由记录器的每个实例独立应用而不是由 X-Ray 服务管理。随着您部署更多主机，固定速率会成倍增加，这使得控制记录的数据量变得更加困难。

开启后 AWS Lambda，您无法修改采样率。如果您的函数由检测服务调用，Lambda 将记录生成由该服务采样的请求的调用。如果启用了主动跟踪且不存在任何跟踪标头，则 Lambda 会做出采样决定。

要配置备份规则，请指示从具有适用于 Node.js 的 X-Ray 开发工具包 `setSamplingRules` 的文件加载采样规则。

Example app.js - 来自文件的采样规则

```
var AWSXRay = require('aws-xray-sdk');
AWSXRay.middleware.setSamplingRules('sampling-rules.json');
```

您也可以在代码中定义规则，并将它们作为对象传递给 `setSamplingRules`。

Example app.js - 来自对象的采样规则

```
var AWSXRay = require('aws-xray-sdk');
var rules = {
  "rules": [ { "description": "Player moves.", "service_name": "*", "http_method": "*",
 "url_path": "/api/move/*", "fixed_target": 0, "rate": 0.05 } ],
  "default": { "fixed_target": 1, "rate": 0.1 },
  "version": 1
  }

AWSXRay.middleware.setSamplingRules(rules);
```

要仅使用本地规则，请调用 `disableCentralizedSampling`。

```
AWSXRay.middleware.disableCentralizedSampling()
```

日志记录

要从开发工具包中记录输出，请调用 `AWSXRay.setLogger(logger)`，其中 `logger` 是提供标准日志记录方法 (`warn`、`info` 等) 的对象。

默认情况下，开发工具包会使用控制台对象上的标准方法将错误消息记录到控制台。可以使用 `AWS_XRAY_DEBUG_MODE` 或 `AWS_XRAY_LOG_LEVEL` 环境变量设置内置记录器的日志级别。有关有效日志级别值的列表，请参阅环境变量。

如果希望为日志提供不同的格式或目标，则可以提供包含您自己的记录器接口实现方式的开发工具包，如下所示。可以使用任何能够实现此接口的对象。这意味着，可以使用 Winton 等许多日志记录库并将其传递给开发工具包。

Example app.js - 日志记录

```
var AWSXRay = require('aws-xray-sdk');
```

```
// Create your own logger, or instantiate one using a library.
var logger = {
  error: (message, meta) => { /* logging code */ },
  warn: (message, meta) => { /* logging code */ },
  info: (message, meta) => { /* logging code */ },
  debug: (message, meta) => { /* logging code */ }
}

AWSXRay.setLogger(logger);
AWSXRay.config([AWSXRay.plugins.EC2Plugin]);
```

在运行其他配置方法之前调用 `setLogger`，确保捕获这些操作的输出。

X-Ray 进程守护程序地址

如果 X-Ray 进程守护程序侦听 `127.0.0.1:2000` 之外的端口或主机，则您可以配置适用于 Node.js 的 X-Ray 开发工具包将跟踪数据发送到不同的 UDP 地址。

```
AWSXRay.setDaemonAddress('host:port');
```

您可以按名称或 IPv4 地址指定主机。

Example app.js - 进程守护程序地址

```
var AWSXRay = require('aws-xray-sdk');
AWSXRay.setDaemonAddress('daemonhost:8082');
```

如果您已将进程守护程序配置为在不同的端口上侦听 TCP 和 UDP，则可以同时在守护程序地址设置中指定二者。

Example app.js – 不同的端口上的进程守护程序地址

```
var AWSXRay = require('aws-xray-sdk');
AWSXRay.setDaemonAddress('tcp:daemonhost:8082 udp:daemonhost:8083');
```

此外，您还可以通过使用 `AWS_XRAY_DAEMON_ADDRESS` 环境变量来设置守护程序地址。

环境变量

您可以使用环境变量来配置适用于 Node.js 的 X-Ray 开发工具包。SDK 支持以下变量。

- AWS_XRAY_CONTEXT_MISSING - 设置为 RUNTIME_ERROR 在您的已检测代码尝试在分段未打开的情况下记录数据时引发异常。

  有效值

  - RUNTIME_ERROR— 引发运行时异常。
  - LOG_ERROR— 记录错误并继续（默认）。
  - IGNORE_ERROR— 忽略错误并继续。

  对于在未打开任何请求时运行的启动代码或者会生成新线程的代码，如果您尝试在其中使用检测过的客户端，则可能发生与缺失分段或子分段相关的错误。
- AWS_XRAY_DAEMON_ADDRESS - 设置 X-Ray 进程守护程序侦听器的主机和端口。默认情况下，SDK 使用用于跟踪数据（UDP）和采样（TCP）的 127.0.0.1:2000。如果您已将进程守护程序配置为侦听不同端口或者进程守护程序在另一台主机上运行，则使用此变量。

  格式

  - 同一个端口 — *address*:*port*
  - 不同的端口 — tcp:*address*:*port* udp:*address*:*port*
- AWS_XRAY_DEBUG_MODE - 设置为 TRUE 以配置开发工具包在 debug 级别将日志输出到控制台。
- AWS_XRAY_LOG_LEVEL - 设置日志记录程序的默认日志级别。有效值为 debug、info、warn、error 和 silent。当设置为时 AWS_XRAY_DEBUG_MODE ，该值将被忽略TRUE。
- AWS_XRAY_TRACING_NAME - 设置 SDK 用于进行分段的服务名称。覆盖您通过 Express 中间件设置的分段名称。

使用适用于 Node.js 的 X-Ray 开发工具包跟踪传入请求

你可以使用适用于 Node.js 的 X-Ray SDK 来跟踪你的 Express 和 Restify 应用程序在亚马逊或亚马逊 ECS 的 EC2 实例上 EC2提供的 AWS Elastic Beanstalk传入 HTTP 请求。

适用于 Node.js 的 X-Ray 开发工具包为使用 Express 和 Restify 框架的应用程序提供中间件。在您将 X-Ray 中间件添加到应用程序时，适用于 Node.js 的 X-Ray 开发工具包会为每个采样请求创建一个分段。此分段包括 HTTP 请求的计时、方法和处置。其他检测会在此分段上创建子分段。

> **Note**
>
> 对于 AWS Lambda 函数，Lambda 会为每个采样请求创建一个分段。请参阅AWS Lambda 和 AWS X-Ray了解更多信息。

每个分段都有一个名称，用于在服务映射中标识您的应用程序。可以静态命名分段，也可以将 SDK 配置为根据传入请求中的主机标头对其进行动态命名。动态命名允许根据请求中的域名对跟踪进行分组，并且在名称不匹配预期模式时（例如，如果主机标头是伪造的）应用默认名称。

> **转发的请求**
>
> 如果负载均衡器或其他中间将请求转发到您的应用程序，X-Ray 会提取请求 `X-Forwarded-For` 标头中的客户端 IP 而非 IP 数据包中的源 IP。由于转发的请求记录的客户端 IP 可以伪造，因此不应信任。

在转发请求时，SDK 在分段中设置附加字段来指示此行为。如果分段包含设置为 `x_forwarded_for` 的字段 `true`，则从 HTTP 请求的 `X-Forwarded-For` 标头获取客户端 IP。

信息处理程序使用包含以下信息的 `http` 数据块为每个传入请求创建一个分段：

- HTTP 方法 - GET、POST、PUT、DELETE 等。
- 客户端地址 - 发送请求的客户端的 IP 地址。
- 响应代码 - 已完成请求的 HTTP 响应代码。
- 时间 - 开始时间（收到请求时）和结束时间（发送响应时）。
- 用户代理 - 请求中的 `user-agent`。
- 内容长度 - 响应中的 `content-length`。

Sections

通过 Express 跟踪传入请求

要使用 Express 中间件，请在定义路由前，先初始化开发工具包客户端并使用 `express.openSegment` 函数返回的中间件。

Example app.js - Express

```
var app = express();

var AWSXRay = require('aws-xray-sdk');
app.use(AWSXRay.express.openSegment('MyApp'));

app.get('/', function (req, res) {
  res.render('index');
});

app.use(AWSXRay.express.closeSegment());
```

定义路由后，按照所示方式使用 `express.closeSegment` 的输出，以便处理适用于 Node.js 的 X-Ray 开发工具包返回的任何错误。

通过 Restify 跟踪传入请求

要使用 Restify 中间件，请初始化开发工具包客户端并运行 `enable`。将您的 Restify 服务器和分段名传递给它。

Example app.js - Restify

```
var AWSXRay = require('aws-xray-sdk');
var AWSXRayRestify = require('aws-xray-sdk-restify');

var restify = require('restify');
var server = restify.createServer();
AWSXRayRestify.enable(server, 'MyApp'));

server.get('/', function (req, res) {
  res.render('index');
});
```

配置分段命名策略

AWS X-Ray 使用服务名称来标识您的应用程序，并将其与您的应用程序使用的其他应用程序、数据库 APIs、外部数据库和 AWS 资源区分开来。当 X-Ray SDK 为传入请求生成分段时，会将应用程序的服务名称记录在分段的名称字段中。

X-Ray SDK 可以用在 HTTP 请求标头中的 hostname 来命名分段。不过，此标头可以伪造，会导致服务地图中出现意料之外的节点。为防止 SDK 由于包含伪造的主机标头的请求而错误地命名分段，必须为传入请求指定一个默认名称。

如果应用程序为多个域的请求提供服务，则可以将 SDK 配置为使用动态命名策略以在分段名称中反映出这一点。动态命名策略允许 SDK 将主机名用于符合预期模式的请求，并将默认名称应用于不符合预期模式的请求。

例如，可能有一款应用程序为发送到三个子域的请求提供服务，分别为 `www.example.com`、`api.example.com` 和 `static.example.com`。可以使用格式 `*.example.com` 的动态命名策略以识别包含不同名称的子域的分段，服务地图上因此会显示三个服务节点。如果应用程序收到包含与该格式不匹配的 hostname 的请求，您将会在服务地图上看到第四个节点，以及您指定的回退名称。

要对所有请求分段使用相同名称，请在初始化中间件时指定应用程序的名称，如前几节所示。

> **Note**
>
> 您可以使用 `AWS_XRAY_TRACING_NAME` 环境变量覆盖您在代码中定义的默认服务名称。

动态命名策略定义一个主机名应匹配的模式和一个在 HTTP 请求中的主机名与该模式不匹配时要使用的默认名称。要动态命名分段，请使用 `AWSXRay.middleware.enableDynamicNaming`。

Example app.js - 动态分段名称

如果请求中的主机名与模式 `*.example.com` 匹配，请使用主机名。否则，请使用 `MyApp`。

```
var app = express();

var AWSXRay = require('aws-xray-sdk');
app.use(AWSXRay.express.openSegment('MyApp'));
AWSXRay.middleware.enableDynamicNaming('*.example.com');
```

```
app.get('/', function (req, res) {
  res.render('index');
});

app.use(AWSXRay.express.closeSegment());
```

使用适用于 Node.js 的 X-Ray SD AWS K 追踪 SDK 调用

当您的应用程序调用 AWS 服务 以存储数据、写入队列或发送通知时，适用于 Node.js 的 X-Ray SDK 会按子分段跟踪下游的调用。被跟踪 AWS 服务的资源以及您在这些服务中访问的资源（例如，Amazon S3 存储桶或 Amazon SQS 队列）在 X-Ray 控制台的跟踪地图上显示为下游节点。

您通过 适用于 JavaScript 的 AWS SDK V2 或适用于 JavaScript 的 AWS SDK V 3 创建的仪器 AWS SDK 客户端。每个 AWS SDK 版本都提供了不同的方法来检测 S AWS DK 客户端。

> **Note**
>
> 目前，与检测 V2 客户端相比， AWS X-Ray 适用于 Node.js 的 SDK 在检测 适用于 JavaScript 的 AWS SDK V3 客户端时返回的细分信息较少。例如，代表对 DynamoDB 的子分段不会返回表名称。如果您在跟踪中需要此区段信息，请考虑使用 适用于 JavaScript 的 AWS SDK V2。

适用于 JavaScript 的 AWS SDK V2

您可以通过在调用中包装 `aws-sdk` require 语句来检测所有 AWS SDK V2 客户端。`AWSXRay.captureAWS`

Example app.js - AWS 开发工具包检测

```
const AWS = AWSXRay.captureAWS(require('aws-sdk'));
```

要检测单个客户端，请将您的 AWS SDK 客户端封装在调用中`AWSXRay.captureAWSClient`。例如，要检测 `AmazonDynamoDB` 客户端：

Example app.js - DynamoDB 客户端检测

```
    const AWSXRay = require('aws-xray-sdk');
...
    const ddb = AWSXRay.captureAWSClient(new AWS.DynamoDB());
```

> **Warning**
>
> 不要将 `captureAWS` 和 `captureAWSClient` 一起使用。这将导致重复的子分段。

如果你想TypeScript与ECMAScript模块 (ESM) 一起使用来加载你的 JavaScript 代码，使用以下示例导入库：

Example app.js- AWS SDK 工具

```
import * as AWS from 'aws-sdk';
import * as AWSXRay from 'aws-xray-sdk';
```

要使用 ESM 检测所有 AWS 客户端，请使用以下代码：

Example app.js- AWS SDK 工具

```
import * as AWS from 'aws-sdk';
import * as AWSXRay from 'aws-xray-sdk';
const XRAY_AWS = AWSXRay.captureAWS(AWS);
const ddb = new XRAY_AWS.DynamoDB();
```

对于所有服务，都可以在 X-Ray 控制台中看到调用的 API 的名称。X-Ray 开发工具包会为一部分服务将信息添加到分段，从而在服务地图中提供更高的粒度。

例如，当使用经过检测的 DynamoDB 客户端发出调用时，对于针对表的调用，开发工具包会将表名称添加到分段中。在控制台中，每个表在服务地图中显示为一个独立的节点，以及没有表作为目标的调用的一般 DynamoDB 节点。

Example 对 DynamoDB 进行调用以保存项目的子分段

```
{
  "id": "24756640c0d0978a",
  "start_time": 1.480305974194E9,
  "end_time": 1.4803059742E9,
  "name": "DynamoDB",
  "namespace": "aws",
  "http": {
    "response": {
      "content_length": 60,
      "status": 200
    }
```

```
  },
  "aws": {
    "table_name": "scorekeep-user",
    "operation": "UpdateItem",
    "request_id": "UBQNSO5AEM8T4FDA4RQDEB94OVTDRVV4K4HIRGVJF66Q9ASUAAJG",
  }
}
```

在您访问指定的资源时，对以下服务的调用会在服务地图中创建额外的节点。没有定向到特定资源的调用，为服务创建了通用节点。

- Amazon DynamoDB - 表名称
- Amazon Simple Storage Service - 存储桶和键名称
- Amazon Simple Queue Service - 队列名称

适用于 JavaScript 的 AWS SDK V3

适用于 JavaScript 的 AWS SDK V3 是模块化的，因此您的代码只加载所需的模块。因此，不可能检测所有 AWS SDK 客户端，因为 V3 不支持该`captureAWS`方法。

如果要 TypeScript 与 ECMAScript 模块 (ESM) 一起使用来加载 JavaScript代码，则可以使用以下示例来导入库：

```
import * as AWS from 'aws-sdk';
import * as AWSXRay from 'aws-xray-sdk';
```

使用`AWSXRay.captureAWSv3Client`方法检测每个 AWS SDK 客户端。例如，要检测 `AmazonDynamoDB` 客户端：

Example app.js - 使用 SDK for Javascript V3 检测 DynamoDB 客户端

```
    const AWSXRay = require('aws-xray-sdk');
    const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
...
    const ddb = AWSXRay.captureAWSv3Client(new DynamoDBClient({ region:
 "region" }));
```

使用 适用于 JavaScript 的 AWS SDK V3 时，当前不会返回表名、存储桶和密钥名称或队列名称等元数据，因此跟踪映射不会像使用 适用于 JavaScript 的 AWS SDK V2 检测 AWS SDK 客户端时那样包含每个命名资源的离散节点。

Example 使用 V3 时调用 DynamoDB 以保存项目的子分段 适用于 JavaScript 的 AWS SDK

```
{
  "id": "24756640c0d0978a",
  "start_time": 1.480305974194E9,
  "end_time": 1.4803059742E9,
  "name": "DynamoDB",
  "namespace": "aws",
  "http": {
    "response": {
      "content_length": 60,
      "status": 200
    }
  },
  "aws": {
    "operation": "UpdateItem",
    "request_id": "UBQNSO5AEM8T4FDA4RQDEB94OVTDRVV4K4HIRGVJF66Q9ASUAAJG",
  }
}
```

使用适用于 Node.js 的 X-Ray 开发工具包跟踪对下游 HTTP Web 服务的调用

当您的应用程序调用微服务或公共 HTTP 时 APIs，您可以使用适用于 Node.js 的 X-Ray SDK 客户端来检测这些调用，并将该 API 作为下游服务添加到服务图中。

将您的 `http` 或 `https` 客户端传递给适用于 Node.js 的 X-Ray 开发工具包的 `captureHTTPs` 方法以跟踪传出调用。

> **Note**
>
> 支持通过 `captureHTTPsGlobal()` API 使用第三方 HTTP 请求库（如 Axios 或 Superagent）进行调用，并且在使用原生 `http` 模块时仍会跟踪它们。

Example app.js - HTTP 客户端

```
var AWSXRay = require('aws-xray-sdk');
var http = AWSXRay.captureHTTPs(require('http'));
```

要在所有 HTTP 客户端上启用跟踪，请先调用 `captureHTTPsGlobal`，然后加载 `http`。

Example app.js - HTTP 客户端（全局）

```
var AWSXRay = require('aws-xray-sdk');
AWSXRay.captureHTTPsGlobal(require('http'));
var http = require('http');
```

当您检测对下游 Web API 的调用时，适用于 Node.js 的 X-Ray 开发工具包记录一个子分段，其中包含有关 HTTP 请求和响应的信息。X-Ray 使用子分段为远程 API 生成推断分段。

Example 下游 HTTP 调用的子分段

```
{
  "id": "004f72be19cddc2a",
  "start_time": 1484786387.131,
  "end_time": 1484786387.501,
  "name": "names.example.com",
  "namespace": "remote",
  "http": {
    "request": {
      "method": "GET",
      "url": "https://names.example.com/"
    },
    "response": {
      "content_length": -1,
      "status": 200
    }
  }
}
```

Example 下游 HTTP 调用的推断分段

```
{
  "id": "168416dc2ea97781",
  "name": "names.example.com",
  "trace_id": "1-62be1272-1b71c4274f39f122afa64eab",
  "start_time": 1484786387.131,
  "end_time": 1484786387.501,
  "parent_id": "004f72be19cddc2a",
  "http": {
    "request": {
```

```
      "method": "GET",
      "url": "https://names.example.com/"
    },
    "response": {
      "content_length": -1,
      "status": 200
    }
  },
  "inferred": true
}
```

使用适用于 Node.js 的 X-Ray 开发工具包跟踪 SQL 查询

通过将 SQL 客户端包含在相应的适用于 Node.js 的 X-Ray 开发工具包客户端方法中来检测 SQL 数据库查询。

- PostgreSQL – `AWSXRay.capturePostgres()`

```
var AWSXRay = require('aws-xray-sdk');
var pg = AWSXRay.capturePostgres(require('pg'));
var client = new pg.Client();
```

- MySQL – `AWSXRay.captureMySQL()`

```
var AWSXRay = require('aws-xray-sdk');
var mysql = AWSXRay.captureMySQL(require('mysql'));
...
var connection = mysql.createConnection(config);
```

在使用检测的客户端发起 SQL 查询时，适用于 Node.js 的 X-Ray 开发工具包会在子分段中记录有关连接和查询的信息。

在 SQL 子段中包括其他数据

您可以向为 SQL 查询生成的子分段添加其他信息，前提是这些子分段已映射到允许列表的 SQL 字段。例如，要在子段中记录经过清理的 SQL 查询字符串，可以将其直接添加到子分段的 SQL 对象中。

Example 将 SQL 分配给子分段

```
    const queryString = 'SELECT * FROM MyTable';
```

```
connection.query(queryString, ...);

// Retrieve the most recently created subsegment
const subs = AWSXRay.getSegment().subsegments;

if (subs & & subs.length > 0) {
  var sqlSub = subs[subs.length - 1];
  sqlSub.sql.sanitized_query = queryString;
}
```

请参阅 AWS X-Ray 开发人员指南中的 SQL 查询，查看加入允许列表的 SQL 字段的完整列表。

使用 X-Ray SDK for Node.js 生成自定义子分段

子分段可为跟踪的分段扩展为了给请求提供服务而已完成的工作的详细信息。每次使用已检测的客户端进行调用时，X-Ray SDK 在子分段中记录生成的信息。您可以创建其他子分段来分组其他子分段，来度量某个代码段的性能如何，或是来记录注释和元数据。

自定义 Express 子分段

使用 `captureAsyncFunc` 函数为调用下游服务的函数创建自定义子分段。

Example app.js - 自定义子分段表示

```
var AWSXRay = require('aws-xray-sdk');

app.use(AWSXRay.express.openSegment('MyApp'));

app.get('/', function (req, res) {
  var host = 'api.example.com';

  AWSXRay.captureAsyncFunc('send', function(subsegment) {
    sendRequest(host, function() {
      console.log('rendering!');
      res.render('index');
      subsegment.close();
    });
  });
});

app.use(AWSXRay.express.closeSegment());
```

```
function sendRequest(host, cb) {
  var options = {
    host: host,
    path: '/',
  };

  var callback = function(response) {
    var str = '';

    response.on('data', function (chunk) {
      str += chunk;
    });

    response.on('end', function () {
      cb();
    });
  }

  http.request(options, callback).end();
};
```

在本示例中，应用程序将创建一个名为 send 的自定义子分段以调用 sendRequest 函数。captureAsyncFunc 传递您在回调函数发出的异步调用完成时必须在回调函数内关闭的子分段。

对于同步函数，您可以使用 captureFunc 函数，这会在函数块完成执行时立即自动结束子分段。

当您在分段或者其他子分段中创建子分段时，X-Ray SDK for Node.js 将为其生成 ID 并记录开始时间和结束时间。

Example 包含元数据的子分段

```
"subsegments": [{
  "id": "6f1605cd8a07cb70",
  "start_time": 1.480305974194E9,
  "end_time": 1.4803059742E9,
  "name": "Custom subsegment for UserModel.saveUser function",
  "metadata": {
    "debug": {
      "test": "Metadata string from UserModel.saveUser"
    }
  },
```

自定义 Lambda 子分段

该 SDK 配置为在检测到运行于 Lambda 中时，将自动创建占位符 Facade 分段。要创建基本子分段（这将在 X-Ray 跟踪地图上创建单个 `AWS::Lambda::Function` 节点），请调用并重新调整 Facade 分段。如果您手动创建具有新 ID 的新分段（同时共享跟踪 ID、父 ID 和采样决策），则可以发送新分段。

Example app.js - 手动自定义子分段

```
const segment = AWSXRay.getSegment(); //returns the facade segment
const subsegment = segment.addNewSubsegment('subseg');
...
subsegment.close();
//the segment is closed by the SDK automatically
```

使用 X-Ray SDK for Node.js，将注释和元数据添加到分段

可以利用注释和元数据记录与请求、环境或应用程序相关的其他信息。可以将注释和元数据添加到 X-Ray 开发工具包创建的分段或您创建的自定义子分段。

注释是带字符串、数字或布尔值的键值对。系统会对注释编制索引，以便与筛选表达式一起使用。使用注释记录要用于对控制台中的跟踪进行分组的数据或在调用 `GetTraceSummaries` API 时使用的数据。

元数据是可以具有任何类型值的键-值对，包括对象和列表，但没有编制索引，无法与筛选条件表达式一起使用。使用元数据记录要存储在跟踪中但不需要用于搜索跟踪的其他数据。

除了注释和元数据之外，您还可以在分段上记录用户 ID 字符串。用户 IDs 被记录在区段的单独字段中，并编制索引以供搜索使用。

Sections

- 使用 X-Ray SDK for Node.js 记录注释
- 使用 X-Ray SDK for Node.js 记录元数据
- 使用适用于 Node.js 的 X-R IDs ay SDK 录制用户

使用 X-Ray SDK for Node.js 记录注释

使用注释记录有关要为其编制索引以进行搜索的分段和子分段的信息。

注释要求

- 键 - X-Ray 注释的键最多可以包含 500 个字母数字字符。除了点或句点（.）之外，不能使用空格或符号
- 值 - X-Ray 注释的值最多可以包含 1,000 个 Unicode 字符。
- 注释的数量 - 每个跟踪最多可使用 50 条注释。

记录注释

1. 获取对当前分段或子分段的引用。

```
var AWSXRay = require('aws-xray-sdk');
...
var document = AWSXRay.getSegment();
```

2. 调用带有字符串键和布尔值、数字值或字符串值的 addAnnotation。

```
document.addAnnotation("mykey", "my value");
```

以下示例说明如何使用包含点和布尔值、数字值或字符串值的字符串键调用 putAnnotation。

```
document.putAnnotation("testkey.test", "my value");
```

开发工具包将注释以键-值对的形式记录在分段文档的 annotations 对象中。使用相同键调用两次 addAnnotation 将覆盖同一分段或子分段上之前记录的值。

要查找具有带特定值的注释的跟踪，请在annotation[*key*]筛选表达式中使用 关键字。

Example app.js - 注释

```
var AWS = require('aws-sdk');
var AWSXRay = require('aws-xray-sdk');
var ddb = AWSXRay.captureAWSClient(new AWS.DynamoDB());
...
app.post('/signup', function(req, res) {
    var item = {
        'email': {'S': req.body.email},
        'name': {'S': req.body.name},
        'preview': {'S': req.body.previewAccess},
```

```
        'theme': {'S': req.body.theme}
    };

    var seg = AWSXRay.getSegment();
    seg.addAnnotation('theme', req.body.theme);

    ddb.putItem({
      'TableName': ddbTable,
      'Item': item,
      'Expected': { email: { Exists: false } }
  }, function(err, data) {
...
```

使用 X-Ray SDK for Node.js 记录元数据

使用元数据记录有关您无需为其编制索引以进行搜索的分段或子分段的信息。元数据值可以是字符串、数字、布尔值或可序列化为 JSON 对象或数组的任何其他对象。

记录元数据

1. 获取对当前分段或子分段的引用。

   ```
   var AWSXRay = require('aws-xray-sdk');
   ...
   var document = AWSXRay.getSegment();
   ```

2. 使用字符串键、布尔值、数字、字符串或对象值以及字符串命名空间调用 addMetadata。

   ```
   document.addMetadata("my key", "my value", "my namespace");
   ```

   或

   调用仅带有键和值的 addMetadata。

   ```
   document.addMetadata("my key", "my value");
   ```

如果您没有指定命名空间，则开发工具包将使用 default。使用相同键调用两次 addMetadata 将覆盖同一分段或子分段上之前记录的值。

使用适用于 Node.js 的 X-R IDs ay SDK 录制用户

记录请求细分中的用户，以识别发送请求的用户。 IDs 此操作与 AWS Lambda 函数不兼容，因为 Lambda 环境中的分段是不可变的。仅可以对分段而不能对子分段应用 `setUser` 调用。

要记录用户 IDs

1. 获取对当前分段或子分段的引用。

```
var AWSXRay = require('aws-xray-sdk');
...
var document = AWSXRay.getSegment();
```

2. 使用发送请求的用户的字符串 ID 调用 `setUser()`。

```
var user = 'john123';

AWSXRay.getSegment().setUser(user);
```

您可以调用 `setUser` 以便在快速应用程序开始处理请求后立即记录用户 ID。如果您只打算使用分段来设置用户 ID，可以在单个行中链接这些调用。

Example app.js - 用户 ID

```
var AWS = require('aws-sdk');
var AWSXRay = require('aws-xray-sdk');
var uuidv4 = require('uuid/v4');
var ddb = AWSXRay.captureAWSClient(new AWS.DynamoDB());
...
    app.post('/signup', function(req, res) {
    var userId = uuidv4();
    var item = {
        'userId': {'S': userId},
        'email': {'S': req.body.email},
        'name': {'S': req.body.name}
    };

    var seg = AWSXRay.getSegment().setUser(userId);

    ddb.putItem({
      'TableName': ddbTable,
```

```
      'Item': item,
      'Expected': { email: { Exists: false } }
  }, function(err, data) {
...
```

要查找用户 ID 的跟踪，请在user筛选表达式中使用 关键字。

使用 Python

有两种方法可用于检测 Python 应用程序，以将跟踪数据发送到 X-Ray：

- AWS OpenTelemetry Python 版 Distro — 提供一组开源库的 AWS 发行版，用于通过 Distro for Collect AWS o r Collector 向多个 AWS 监控解决方案（包括亚马逊和亚马逊 OpenSearch 服务）发送相关的指标和跟踪。 CloudWatch AWS X-Ray OpenTelemetry
- AWS X-Ray 适用于 Python 的 SDK — 一组库，用于通过 X-Ray 守护程序生成跟踪并将其发送到 X-Ray。

有关更多信息，请参阅 在 AWS Distro for 和 X-Ray OpenTelemetry 之间进行选择 SDKs。

AWS 适用于 Python 的 OpenTelemetry 发行版

使用适用于 OpenTelemetry (ADOT) Python 的 AWS Distro，您可以对应用程序进行一次检测，然后将相关的指标和跟踪发送到多个 AWS 监控解决方案 CloudWatch，包括亚马逊 AWS X-Ray和亚马逊服务。 OpenSearch 将 X-Ray 与 ADOT 配合使用需要两个组件：启用 OpenTelemetry SDK 以与 X-Ray 配合使用，以及启用 Collecto OpenTelemetryr 的AWS Distro 以与 X-Ray 配合使用。ADOT Python 支持自动检测，使您的应用程序无需更改代码即可发送跟踪信息。

要开始使用，请参阅 OpenTelemetry Python AWS 发行版文档。

有关将 Distro 用于 with AWS X-Ray 和 other 的 OpenTelemetry 更多信息 AWS 服务，请参阅 AWS Distro for OpenTelemetry 或 AWS Distro for AWS D ocuments。 OpenTelemetry

有关语言支持和用法的更多信息，请参阅上的 O AWS bservability。 GitHub

AWS X-Ray Python 软件开发工具包

X-Ray SDK for Python 是一个面向 Python Web 应用程序的库，该库提供用于生成跟踪数据并将其发送到 X-Ray 进程守护程序的类和方法。跟踪数据包括有关应用程序处理的传入 HTTP 请求的信息，以及应用程序使用 AWS SDK、HTTP 客户端或 SQL 数据库连接器对下游服务进行的调用的信息。您还可以手动创建分段并在注释和元数据中添加调试信息。

可以使用 `pip` 下载 SDK。

```
$ pip install aws-xray-sdk
```

> **Note**
>
> X-Ray SDK for Python 是一个开源项目。你可以关注该项目并在 github 上 GitHub提交议题和拉取请求。 com/aws/aws-xray-sdk-python

如果您使用的是 Django 或 Flask，请首先将 SDK 中间件添加到您的应用程序以跟踪传入请求。此中间件为每个被跟踪请求创建一个分段并在发送响应时完成该分段。当分段打开时，您可以使用开发工具包客户端的方法将信息添加到分段，并创建子分段以跟踪下游调用。开发工具包还会自动记录在分段打开时应用程序引发的异常。对于其他应用程序，您可以手动创建分段。

对于由经过检测的应用程序或服务调用的 Lambda 函数，Lambda 会读取跟踪标头并自动跟踪采样的请求。对于其他函数，您可以将 Lambda 配置为采样和跟踪传入请求。无论哪种情况，Lambda 都会创建分段并将其提供给 X-Ray 开发工具包。

> **Note**
>
> 在 Lambda 上，X-Ray 开发工具包是可选的。如果您不在函数中使用它，您的服务映射仍将包含一个用于 Lambda 服务的节点，以及每个 Lambda 函数的节点。可通过添加该开发工具包检测函数代码，将子分段添加到 Lambda 记录的函数分段。请参阅AWS Lambda 和 AWS X-Ray了解更多信息。

有关在 Lambda 中检测过的示例 Python 函数，请参阅工作线程。

接下来，通过修补应用程序库，使用 X-Ray SDK for Python 检测下游调用。该 SDK 支持以下库。

支持的库

- `botocore`，`boto3`— 仪器 AWS SDK for Python (Boto) 客户。
- `pynamodb` - 检测 Amazon DynamoDB 客户端的 PynamoDB 版本。
- `aiobotocore`、`aioboto3` - 检测 SDK for Python 客户端的 asyncio 集成版本。
- `requests`、`aiohttp` - 检测高级别 HTTP 客户端。
- `httplib`、`http.client` - 检测低级别 HTTP 客户端和使用这些客户端的更高级别的库。
- `sqlite3`— 仪器 SQLite 客户。
- `mysql-connector-python` - 检测 MySQL 客户端。
- `pg8000` - 检测 Pure-Python PostgreSQL 接口。

- psycopg2 - 检测 PostgreSQL 数据库适配器。
- pymongo - 检测 MongoDB 客户端。
- pymysql— 针对 My PyMy SQL 和 MariaDB 的基于 SQL 的客户端。

每当您的应用程序调用 SQL 数据库或其他 HTTP 服务时，SDK 都会在子分段中记录有关该调用的信息。 AWS AWS 服务 您在服务中访问的资源将作为下游节点显示在跟踪地图上，以帮助您识别各个连接上的错误和限制问题。

在开始使用 SDK 后，通过配置记录器和中间件来自定义其行为。您可以添加插件来记录有关应用程序上运行的计算资源的数据，通过定义采样规则来自定义采样行为，设置日志级别以在应用程序日志中查看来自开发工具包的更多或更少的信息。

记录有关请求以及应用程序在注释和元数据中所做的工作的其他信息。注释是简单的键值对，已为这些键值对编制索引以用于筛选条件表达式，以便您能够搜索包含特定数据的跟踪。元数据条目的限制性较低，并且可以记录整个对象和数组 - 可序列化为 JSON 的任何项目。

> **注释和元数据**
>
> 注释和元数据是您使用 X-Ray 开发工具包添加到分段的任意文本。系统会对注释编制索引，以便与筛选表达式一起使用。元数据未编制索引，但可以使用 X-Ray 控制台或 API 在原始分段中查看。您授予 X-Ray 读取权限的任何人都可以查看这些数据。

当代码中具有大量检测的客户端时，一个请求分段可包含大量子分段，检测的客户端发起的每个调用均对应一个子分段。您可以通过将客户端调用包含在自定义子分段中来整理子分段并为其分组。您可以为整个函数或任何代码部分创建自定义子分段。然后，您可以在子分段上记录元数据和注释，而不必在父分段上写入所有内容。

有关 SDK 的类和方法的参考文档，请参阅 AWS X-Ray SDK for Python API 参考。

要求

X-Ray SDK for Python 支持以下语言和库版本。

- Python - 2.7、3.4 和更新版本
- Django - 1.10 和更新版本
- Flask - 0.10 和更新版本

- aiohttp - 2.3.0 和更新版本
- AWS SDK for Python (Boto) - 1.4.0 和更新版本
- botocore - 1.5.0 和更新版本
- enum — 0.4.7 和更高版本，适用于 Python 版本 3.4.0 及更高版本
- jsonpickle — 1.0.0 和更新版本
- setuptools — 40.6.3 和更新版本
- wrapt - 1.11.0 和更新版本

依赖关系管理

可从 `pip` 获得 X-Ray SDK for Python。

- 程序包 - `aws-xray-sdk`

在您的 `requirements.txt` 文件中添加 SDK 作为依赖项。

Example requirements.txt

```
aws-xray-sdk==2.4.2
boto3==1.4.4
botocore==1.5.55
Django==1.11.3
```

如果您使用 Elastic Beanstalk 部署您的应用程序，Elastic Beanstalk 会自动安装 `requirements.txt` 中的所有程序包。

配置适用于 Python 的 X-Ray 开发工具包

适用于 Python 的 X-Ray 开发工具包具有提供全局记录器的、名为 `xray_recorder` 的类。您可以配置全局记录器以自定义为传入 HTTP 调用创建分段的中间件。

Sections

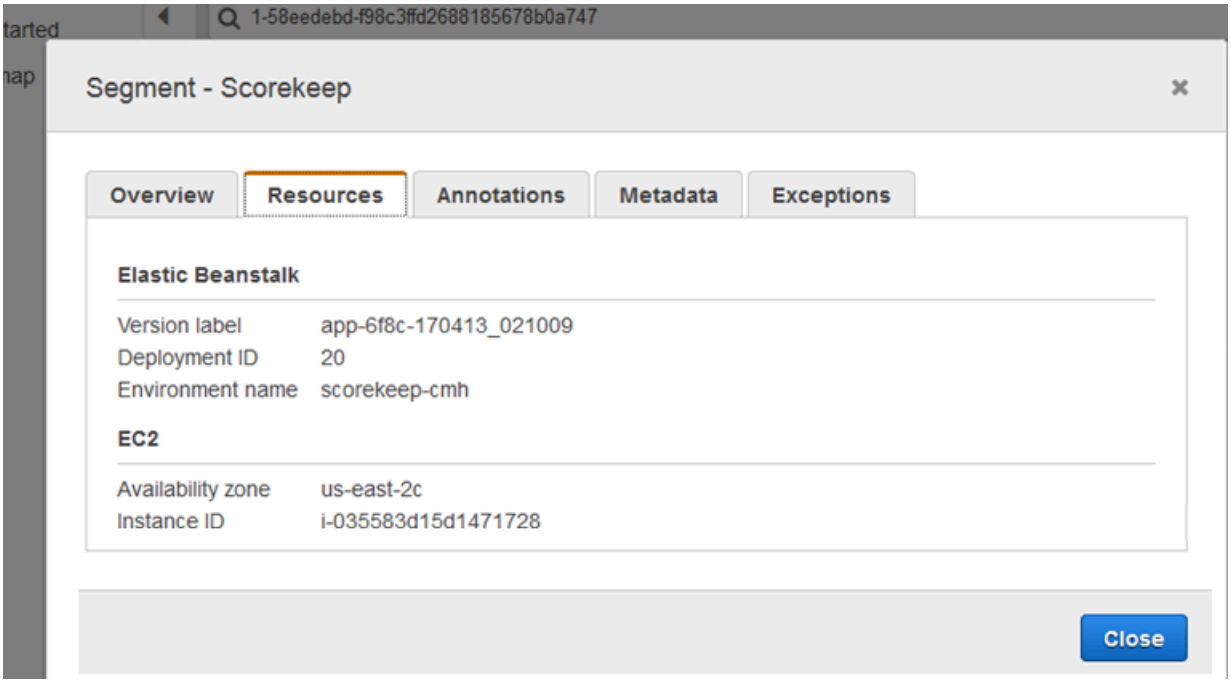
- 代码中的记录器配置
- 使用 Django 时的记录器配置
- 环境变量

服务插件

`plugins` 用于记录有关托管应用程序的服务的信息。

插件

- Amazon EC2 — `EC2Plugin` 添加实例 ID、可用区和 CloudWatch 日志组。
- Elastic Beanstalk - `ElasticBeanstalkPlugin` 添加环境名称、版本标签和部署 ID。
- Amazon ECS — `ECSPlugin` 添加容器 ID。



要使用插件，请在 `xray_recorder` 上调用 `configure`。

```
from aws_xray_sdk.core import xray_recorder
from aws_xray_sdk.core import patch_all

xray_recorder.configure(service='My app')
plugins = ('ElasticBeanstalkPlugin', 'EC2Plugin')
xray_recorder.configure(plugins=plugins)
```

```
patch_all()
```

> **Note**
>
> 由于 `plugins` 是作为元组传入的，因此请确保包含指定单一插件尾随的 `,`。例如，`plugins = ('EC2Plugin',)`

您还可以使用环境变量来配置记录器，它优先于在代码中设置的值。

配置插件，然后再修补库，从而记录下游调用。

该 SDK 还使用插件设置为设置分段上的 `origin` 字段。这表示运行您的应用程序的 AWS 资源类型。当您使用多个插件时，SDK 使用以下解析顺序来确定来源： ElasticBeanstalk > EKS > ECS > EC2。

采样规则

该 SDK 使用您在 X-Ray 控制台中定义的采样规则来确定要记录的请求。默认规则跟踪每秒的第一个请求，以及所有将跟踪发送到 X-Ray 的服务的任何其他请求的百分之五。在 X-Ray 控制台中创建其他规则以自定义为每个应用程序记录的数据量。

该 SDK 按照定义的顺序应用自定义规则。如果请求与多个自定义规则匹配，则 SDK 仅应用第一条规则。

> **Note**
>
> 如果 SDK 无法访问 X-Ray 来获取采样规则，它将恢复为默认的本地规则，即每秒第一个请求以及每个主机所有其他请求的百分之五。如果主机无权调用采样，或者无法连接到 X-Ray 守护程序 APIs，后者充当 SDK 发出的 API 调用的 TCP 代理，则可能会发生这种情况。

您还可以将 SDK 配置为从 JSON 文档加载采样规则。在 X-Ray 采样不可用的情况下，SDK 可以使用本地规则作为备份，也可以只使用本地规则。

Example sampling-rules.json

```
{
  "version": 2,
  "rules": [
```

```
    {
      "description": "Player moves.",
      "host": "*",
      "http_method": "*",
      "url_path": "/api/move/*",
      "fixed_target": 0,
      "rate": 0.05
    }
  ],
  "default": {
    "fixed_target": 1,
    "rate": 0.1
  }
}
```

此示例定义了一个自定义规则和一个默认规则。自定义规则采用百分之五的采样率，对于 `/api/move/` 之下的路径要跟踪的请求数量不设下限。默认规则中每秒的第一个请求以及其他请求的百分之十。

在本地定义规则的缺点是，固定目标由记录器的每个实例独立应用而不是由 X-Ray 服务管理。随着您部署更多主机，固定速率会成倍增加，这使得控制记录的数据量变得更加困难。

开启后 AWS Lambda，您无法修改采样率。如果您的函数由检测服务调用，Lambda 将记录生成由该服务采样的请求的调用。如果启用了活动跟踪且不存在任何跟踪标头，则 Lambda 会做出采样决定。

要配置备份采样规则，请调用`xray_recorder.configure`，如以下示例所示，其中要么*`rules`*是规则字典，要么是包含采样规则的 JSON 文件的绝对路径。

```
xray_recorder.configure(sampling_rules=rules)
```

要仅使用本地规则，请使用 `LocalSampler` 配置记录器。

```
from aws_xray_sdk.core.sampling.local.sampler import LocalSampler
xray_recorder.configure(sampler=LocalSampler())
```

您还可以配置全局记录器，以禁止对所有传入请求进行采样和检测。

Example main.py - 禁用采样

```
xray_recorder.configure(sampling=False)
```

日志记录

开发工具包使用 Python 内置的 `logging` 模块，其中包含默认 `WARNING` 日志记录级别。获取对 `aws_xray_sdk` 类的日志记录器的引用，并对其调用 `setLevel` 来为库以及应用程序的其余部分配置不同的日志级别。

Example app.py - 日志记录

```
logging.basicConfig(level='WARNING')
logging.getLogger('aws_xray_sdk').setLevel(logging.ERROR)
```

当您手动生成子分段时，使用调试日志来识别诸如未结束子分段之类的问题。

代码中的记录器配置

其他设置包含在 `xray_recorder` 的 `configure` 方法中。

- `context_missing` - 设置为 `LOG_ERROR` 可避免在您的已检测代码尝试在分段未打开的情况下记录数据时引发异常。
- `daemon_address` - 设置 X-Ray 进程守护程序侦听器的主机和端口。
- `service` - 设置开发工具包用于进行分段的服务名称。
- `plugins`— 记录有关您的应用程序 AWS 资源的信息。
- `sampling` 设置为 `False` 可禁用采样。
- `sampling_rules` - 设置包含您的采样规则的 JSON 文件的路径。

Example main.py - 禁用缺少上下文异常

```
from aws_xray_sdk.core import xray_recorder

xray_recorder.configure(context_missing='LOG_ERROR')
```

使用 Django 时的记录器配置

如果您使用 Django 框架，您可以使用 Django `settings.py` 文件来配置全局记录器的选项。

- `AUTO_INSTRUMENT`（仅限 Django） - 记录内置数据库和模板渲染操作的子分段。
- `AWS_XRAY_CONTEXT_MISSING` - 设置为 `LOG_ERROR` 可避免在您的已检测代码尝试在分段未打开的情况下记录数据时引发异常。

- AWS_XRAY_DAEMON_ADDRESS - 设置 X-Ray 进程守护程序侦听器的主机和端口。
- AWS_XRAY_TRACING_NAME - 设置开发工具包用于进行分段的服务名称。
- PLUGINS— 记录有关您的应用程序 AWS 资源的信息。
- SAMPLING 设置为 False 可禁用采样。
- SAMPLING_RULES - 设置包含您的采样规则的 JSON 文件的路径。

要启用 settings.py 中的记录器配置，请将 Django 中间件添加到已安装应用程序列表中。

Example settings.py - 已安装的应用程序

```
INSTALLED_APPS = [
    ...
    'django.contrib.sessions',
    'aws_xray_sdk.ext.django',
]
```

在名为 XRAY_RECORDER 的 dict 中配置可用设置。

Example settings.py - 已安装的应用程序

```
XRAY_RECORDER = {
    'AUTO_INSTRUMENT': True,
    'AWS_XRAY_CONTEXT_MISSING': 'LOG_ERROR',
    'AWS_XRAY_DAEMON_ADDRESS': '127.0.0.1:5000',
    'AWS_XRAY_TRACING_NAME': 'My application',
    'PLUGINS': ('ElasticBeanstalkPlugin', 'EC2Plugin', 'ECSPlugin'),
    'SAMPLING': False,
}
```

环境变量

您可以使用环境变量配置适用于 Python 的 X-Ray 开发工具包。开发工具包支持以下变量：

- AWS_XRAY_TRACING_NAME - 设置 SDK 用于进行分段的服务名称。覆盖您以编程方式设置的服务名称。
- AWS_XRAY_SDK_ENABLED - 如果设置为 false，则会禁用开发工具包。默认情况下，开发工具包处于启用状态，除非环境变量设置为 false。

  - 禁用时，全局记录器会自动生成不发送到进程守护程序的虚拟分段和子分段，并禁用自动修补。中间件是作为全局记录器的包装器编写的。通过中间件生成的所有分段和子分段也成为虚拟分段和虚拟子分段。
  - 通过环境变量设置 AWS_XRAY_SDK_ENABLED 值，或者通过与 aws_xray_sdk 库中 global_sdk_config 对象的直接交互来设置。环境变量的设置会覆盖这些交互。
- AWS_XRAY_DAEMON_ADDRESS - 设置 X-Ray 进程守护程序侦听器的主机和端口。默认情况下，SDK 使用用于跟踪数据（UDP）和采样（TCP）的 127.0.0.1:2000。如果您已将进程守护程序配置为侦听不同端口或者进程守护程序在另一台主机上运行，则使用此变量。

  格式

  - 同一个端口 — *address*:*port*
  - 不同的端口 — tcp:*address*:*port* udp:*address*:*port*
- AWS_XRAY_CONTEXT_MISSING - 设置为 RUNTIME_ERROR 在您的已检测代码尝试在分段未打开的情况下记录数据时引发异常。

  有效值

  - RUNTIME_ERROR— 引发运行时异常。
  - LOG_ERROR— 记录错误并继续（默认）。
  - IGNORE_ERROR— 忽略错误并继续。

  对于在未打开任何请求时运行的启动代码或者会生成新线程的代码，如果您尝试在其中使用检测过的客户端，则可能发生与缺失分段或子分段相关的错误。

环境变量覆盖在代码中设置的值。

使用适用于 Python 中间件的 X-Ray 开发工具包跟踪传入请求

在您将中间件添加到应用程序并配置分段名称时，适用于 Python 的 X-Ray 开发工具包会为每个采样请求创建一个分段。此分段包括 HTTP 请求的计时、方法和处置。其他检测会在此分段上创建子分段。

适用于 Python 的 X-Ray 开发工具包支持以下中间件来检测传入的 HTTP 请求：

- Django
- Flask
- Bottle

> **Note**
>
> 对于 AWS Lambda 函数，Lambda 会为每个采样请求创建一个分段。请参阅AWS Lambda 和 AWS X-Ray了解更多信息。

有关在 Lambda 中检测过的示例 Python 函数，请参阅工作线程。

对于其他框架上的脚本或 Python 应用程序，您可以手动创建分段。

每个分段都有一个名称，用于在服务映射中标识您的应用程序。可以静态命名分段，也可以将 SDK 配置为根据传入请求中的主机标头对其进行动态命名。动态命名允许根据请求中的域名对跟踪进行分组，并且在名称不匹配预期模式时（例如，如果主机标头是伪造的）应用默认名称。

> **转发的请求**
>
> 如果负载均衡器或其他中间将请求转发到您的应用程序，X-Ray 会提取请求 `X-Forwarded-For` 标头中的客户端 IP 而非 IP 数据包中的源 IP。由于转发的请求记录的客户端 IP 可以伪造，因此不应信任。

在转发请求时，SDK 在分段中设置附加字段来指示此行为。如果分段包含设置为 `x_forwarded_for` 的字段 `true`，则从 HTTP 请求的 `X-Forwarded-For` 标头获取客户端 IP。

中间件使用包含以下信息的 `http` 块为每个传入请求创建一个分段：

- HTTP 方法 - GET、POST、PUT、DELETE 等。
- 客户端地址 - 发送请求的客户端的 IP 地址。
- 响应代码 - 已完成请求的 HTTP 响应代码。
- 时间 - 开始时间（收到请求时）和结束时间（发送响应时）。
- 用户代理 - 请求中的 `user-agent`。
- 内容长度 - 响应中的 `content-length`。

Sections

- 将中间件添加到应用程序 (Django)
- 将中间件添加到应用程序 (Flask)
- 将中间件添加到应用程序 (Bottle)

- 手动检测 Python 代码
- 配置分段命名策略

将中间件添加到应用程序 (Django)

将中间件添加到 MIDDLEWARE 文件中的 settings.py 列表。X-Ray 中间件应位于 settings.py 文件中的第一行，以确保在其他中间件失败的请求得到记录。

Example settings.py - 适用于 Python 中间件的 X-Ray 开发工具包

```
MIDDLEWARE = [
    'aws_xray_sdk.ext.django.middleware.XRayMiddleware',
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware'
]
```

将 X-Ray 开发工具包 Django 添加到 settings.py 文件中的 INSTALLED_APPS 列表。这将允许在应用程序启动期间配置 X-Ray 记录器。

Example settings.py - 适用于 Python Django 应用的 X-Ray 开发工具包

```
INSTALLED_APPS = [
    'aws_xray_sdk.ext.django',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

在 settings.py 文件中配置分段名称。

Example settings.py - 分段名称

```
XRAY_RECORDER = {
```

```
    'AWS_XRAY_TRACING_NAME': 'My application',
    'PLUGINS': ('EC2Plugin',),
}
```

这告知 X-Ray 记录器使用默认采样率跟踪您的 Django 应用程序处理的请求。您可以在您的 Django 设置文件中配置记录器，以便应用自定义采样规则或更改其他设置。

> **Note**
>
> 由于 `plugins` 是作为元组传入的，因此请确保包含指定单一插件尾随的 `,`。例如，`plugins = ('EC2Plugin',)`

将中间件添加到应用程序 (Flask)

要检测您的 Flask 应用程序，请先在 `xray_recorder` 上配置分段名称。然后，在代码中使用 `XRayMiddleware` 函数来修补您的 Flask 应用程序。

Example app.py

```
from aws_xray_sdk.core import xray_recorder
from aws_xray_sdk.ext.flask.middleware import XRayMiddleware

app = Flask(__name__)

xray_recorder.configure(service='My application')
XRayMiddleware(app, xray_recorder)
```

这告知 X-Ray 记录器使用默认采样率跟踪您的 Flask 应用程序处理的请求。您可以在代码中配置记录器，以便应用自定义采样规则或更改其他设置。

将中间件添加到应用程序 (Bottle)

要检测您的 Bottle 应用程序，请先在 `xray_recorder` 上配置分段名称。然后，在代码中使用 `XRayMiddleware` 函数来修补您的 Bottle 应用程序。

Example app.py

```
from aws_xray_sdk.core import xray_recorder
from aws_xray_sdk.ext.bottle.middleware import XRayMiddleware

app = Bottle()
```

```
xray_recorder.configure(service='fallback_name', dynamic_naming='My application')
app.install(XRayMiddleware(xray_recorder))
```

这告知 X-Ray 记录器使用默认采样率跟踪您的 Bottle 应用程序处理的请求。您可以在代码中配置记录器，以便应用自定义采样规则或更改其他设置。

手动检测 Python 代码

如果您不使用 Django 或 Flask，则可以手动创建分段。您可以为每个传入的请求创建区段，也可以围绕已修补的 HTTP 或 AWS SDK 客户端创建区段，为录制器添加子分段提供上下文。

Example main.py - 手动检测

```
from aws_xray_sdk.core import xray_recorder

# Start a segment
segment = xray_recorder.begin_segment('segment_name')
# Start a subsegment
subsegment = xray_recorder.begin_subsegment('subsegment_name')

# Add metadata and annotations
segment.put_metadata('key', dict, 'namespace')
subsegment.put_annotation('key', 'value')

# Close the subsegment and segment
xray_recorder.end_subsegment()
xray_recorder.end_segment()
```

配置分段命名策略

AWS X-Ray 使用服务名称来标识您的应用程序，并将其与您的应用程序使用的其他应用程序、数据库 APIs、外部数据库和 AWS 资源区分开来。当 X-Ray SDK 为传入请求生成分段时，会将应用程序的服务名称记录在分段的名称字段中。

X-Ray SDK 可以用在 HTTP 请求标头中的 hostname 来命名分段。不过，此标头可以伪造，会导致服务地图中出现意料之外的节点。为防止 SDK 由于包含伪造的主机标头的请求而错误地命名分段，必须为传入请求指定一个默认名称。

如果应用程序为多个域的请求提供服务，则可以将 SDK 配置为使用动态命名策略以在分段名称中反映出这一点。动态命名策略允许 SDK 将主机名用于符合预期模式的请求，并将默认名称应用于不符合预期模式的请求。

例如，可能有一款应用程序为发送到三个子域的请求提供服务，分别为 `www.example.com`、`api.example.com` 和 `static.example.com`。可以使用格式 `*.example.com` 的动态命名策略以识别包含不同名称的子域的分段，服务地图上因此会显示三个服务节点。如果应用程序收到包含与该格式不匹配的 hostname 的请求，您将会在服务地图上看到第四个节点，以及您指定的回退名称。

要对所有请求分段使用相同名称，请在配置记录器时指定应用程序的名称，如前几节所示。

动态命名策略定义一个主机名应匹配的模式和一个在 HTTP 请求中的主机名与该模式不匹配时要使用的默认名称。要在 Django 中动态命名分段，请将 `DYNAMIC_NAMING` 设置添加到您的 settings.py 文件。

Example settings.py - 动态命名

```
XRAY_RECORDER = {
    'AUTO_INSTRUMENT': True,
    'AWS_XRAY_TRACING_NAME': 'My application',
    'DYNAMIC_NAMING': '*.example.com',
    'PLUGINS': ('ElasticBeanstalkPlugin', 'EC2Plugin')
}
```

您可以在模式中使用“*”来匹配任何字符串，或使用“?”来匹配任意单个字符。对于 Flask，在代码中配置记录器。

Example main.py - 分段名称

```
from aws_xray_sdk.core import xray_recorder
xray_recorder.configure(service='My application')
xray_recorder.configure(dynamic_naming='*.example.com')
```

> **Note**
>
> 您可以使用 `AWS_XRAY_TRACING_NAME` 环境变量覆盖您在代码中定义的默认服务名称。

修补库以检测下游调用

要检测下游调用，请使用适用于 Pyhon 的 X-Ray 开发工具包修补您的应用程序使用的库。适用于 Python 的 X-Ray 开发工具包可以修补以下库。

支持的库

- botocore，boto3— 仪器 AWS SDK for Python (Boto) 客户。
- pynamodb - 检测 Amazon DynamoDB 客户端的 PynamoDB 版本。
- aiobotocore、aioboto3 - 检测 SDK for Python 客户端的 asyncio 集成版本。
- requests、aiohttp - 检测高级别 HTTP 客户端。
- httplib、http.client - 检测低级别 HTTP 客户端和使用这些客户端的更高级别的库。
- sqlite3— 仪器 SQLite 客户。
- mysql-connector-python - 检测 MySQL 客户端。
- pg8000 - 检测 Pure-Python PostgreSQL 接口。
- psycopg2 - 检测 PostgreSQL 数据库适配器。
- pymongo - 检测 MongoDB 客户端。
- pymysql— 针对 My PyMy SQL 和 MariaDB 的基于 SQL 的客户端。

如果您使用已修补的库，适用于 Pyhon 的 X-Ray 开发工具包会为调用创建子分段，并记录请求和响应中的信息。必须通过开发工具包中间件或 AWS Lambda提供分段，以供开发工具包创建子分段。

> **Note**
>
> 如果您使用 SQLAlchemy ORM，则可以通过导入 SDK 版本的会话和查询类 SQLAlchemy来检测 SQL 查询。有关说明，请参阅使用 SQLAlchemy ORM。

要修补所有可用的库，请使用 `aws_xray_sdk.core` 中的 `patch_all` 函数。某些库（例如 `httplib` 和 `urllib`）可能需要通过调用 `patch_all(double_patch=True)` 启用双重修补。

Example main.py - 修补所有支持的库

```
import boto3
import botocore
import requests
import sqlite3

from aws_xray_sdk.core import xray_recorder
from aws_xray_sdk.core import patch_all

patch_all()
```

要修补单个库，请使用库名称的元组调用 patch。为此，您需要提供单个元素列表。

Example main.py - 修补特定的库

```
import boto3
import botocore
import requests
import mysql-connector-python

from aws_xray_sdk.core import xray_recorder
from aws_xray_sdk.core import patch

libraries = ('botocore')
patch(libraries)
```

> **Note**
>
> 在某些情况下，用于修补库的键与库名称不匹配。有些键可作为一个或多个库的别名。
>
> 库别名
>
> - httplib – httplib 和 http.client
> - mysql – mysql-connector-python

跟踪异步工作的上下文

对于集成了 asyncio 的库，或者要为异步函数创建子分段，您还必须使用异步上下文配置适用于 Pyhon 的 X-Ray 开发工具包。导入 AsyncContext 类，并将它的一个实例传递到 X-Ray 记录器。

> **Note**
>
> Web 框架支持库（例如 AIOHTTP）不未通过 aws_xray_sdk.core.patcher 模块处理。它们将不会出现在支持的库的 patcher 目录中。

Example main.py - 修补程序 aioboto3

```
import asyncio
import aioboto3
import requests
```

```
from aws_xray_sdk.core.async_context import AsyncContext
from aws_xray_sdk.core import xray_recorder
xray_recorder.configure(service='my_service', context=AsyncContext())
from aws_xray_sdk.core import patch

libraries = (['aioboto3'])
patch(libraries)
```

使用 AWS 适用于 Python 的 X-Ray 软件开发工具包追踪 SDK

当您的应用程序调用 AWS 服务 以存储数据、写入队列或发送通知时，适用于 Python 的 X-Ray SDK 会在子分段中跟踪下游的调用。在这些服务（例如，Amazon S3 存储桶或 Amazon SQS 队列）中追踪的资源 AWS 服务 和访问的资源在 X-Ray 控制台的跟踪地图上显示为下游节点。

当你修补botocore库时，适用于 Python 的 X-Ray AWS SDK 会自动检测所有 SDK 客户端。您无法检测单个客户端。

对于所有服务，都可以在 X-Ray 控制台中看到调用的 API 的名称。X-Ray 开发工具包会为一部分服务将信息添加到分段，从而在服务地图中提供更高的粒度。

例如，当使用经过检测的 DynamoDB 客户端发出调用时，对于针对表的调用，开发工具包会将表名称添加到分段中。在控制台中，每个表在服务地图中显示为一个独立的节点，以及没有表作为目标的调用的一般 DynamoDB 节点。

Example 对 DynamoDB 进行调用以保存项目的子分段

```
{
  "id": "24756640c0d0978a",
  "start_time": 1.480305974194E9,
  "end_time": 1.4803059742E9,
  "name": "DynamoDB",
  "namespace": "aws",
  "http": {
    "response": {
      "content_length": 60,
      "status": 200
    }
  },
  "aws": {
    "table_name": "scorekeep-user",
    "operation": "UpdateItem",
    "request_id": "UBQNSO5AEM8T4FDA4RQDEB94OVTDRVV4K4HIRGVJF66Q9ASUAAJG",
```

```
  }
}
```

在您访问指定的资源时，对以下服务的调用会在服务地图中创建额外的节点。没有定向到特定资源的调用，为服务创建了通用节点。

- Amazon DynamoDB - 表名称
- Amazon Simple Storage Service - 存储桶和键名称
- Amazon Simple Queue Service - 队列名称

使用适用于 Python 的 X-Ray 开发工具包跟踪对下游 HTTP Web 服务的调用

当您的应用程序调用微服务或公共 HTTP 时 APIs，您可以使用适用于 Python 的 X-Ray SDK 来检测这些调用，并将该 API 作为下游服务添加到服务图中。

要检测 HTTP 客户端，请修补用于进行传出调用的库。如果您使用 `requests` 或 Python 的内置 HTTP 客户端，您只需进行上述操作。对于 `aiohttp`，还需为记录器配置异步上下文。

如果您使用的是 `aiohttp` 3 的客户端 API，您还需要使用软件开发工具包提供的跟踪配置实例来配置 `ClientSession` 的客户端 API。

Example **`aiohttp`** 3 客户端 API

```
from aws_xray_sdk.ext.aiohttp.client import aws_xray_trace_config

async def foo():
    trace_config = aws_xray_trace_config()
    async with ClientSession(loop=loop, trace_configs=[trace_config]) as session:
        async with session.get(url) as resp
            await resp.read()
```

当您检测对下游 Web API 的调用时，适用于 Python 的 X-Ray 开发工具包记录一个子分段，其中包含有关 HTTP 请求和响应的信息。X-Ray 使用子分段为远程 API 生成推断分段。

Example 下游 HTTP 调用的子分段

```
{
  "id": "004f72be19cddc2a",
  "start_time": 1484786387.131,
  "end_time": 1484786387.501,
  "name": "names.example.com",
```

```
  "namespace": "remote",
  "http": {
    "request": {
      "method": "GET",
      "url": "https://names.example.com/"
    },
    "response": {
      "content_length": -1,
      "status": 200
    }
  }
}
```

Example 下游 HTTP 调用的推断分段

```
{
  "id": "168416dc2ea97781",
  "name": "names.example.com",
  "trace_id": "1-62be1272-1b71c4274f39f122afa64eab",
  "start_time": 1484786387.131,
  "end_time": 1484786387.501,
  "parent_id": "004f72be19cddc2a",
  "http": {
    "request": {
      "method": "GET",
      "url": "https://names.example.com/"
    },
    "response": {
      "content_length": -1,
      "status": 200
    }
  },
  "inferred": true
}
```

使用适用于 Python 的 X-Ray 开发工具包生成自定义子分段

子分段可为跟踪的分段扩展为了给请求提供服务而已完成的工作的详细信息。每次使用已检测的客户端进行调用时，X-Ray SDK 在子分段中记录生成的信息。您可以创建其他子分段来分组其他子分段，来度量某个代码段的性能如何，或是来记录注释和元数据。

要管理子分段，请使用 `begin_subsegment` 和 `end_subsegment` 方法。

Example main.py - 自定义子分段

```
from aws_xray_sdk.core import xray_recorder

subsegment = xray_recorder.begin_subsegment('annotations')
subsegment.put_annotation('id', 12345)
xray_recorder.end_subsegment()
```

要为同步函数创建子分段，请使用 @xray_recorder.capture 装饰器。您可以将子分段名称传递到捕获函数，或者省略以使用函数名称。

Example main.py - 函数子分段

```
from aws_xray_sdk.core import xray_recorder

@xray_recorder.capture('## create_user')
def create_user():
...
```

对于异步函数，请使用 @xray_recorder.capture_async 装饰器，并将异步上下文传递到记录器。

Example main.py - 异步函数子分段

```
from aws_xray_sdk.core.async_context import AsyncContext
from aws_xray_sdk.core import xray_recorder
xray_recorder.configure(service='my_service', context=AsyncContext())

@xray_recorder.capture_async('## create_user')
async def create_user():
    ...

async def main():
    await myfunc()
```

当您在分段或者其他子分段中创建子分段时，适用于 Python 的 X-Ray 开发工具包会为其生成 ID 并记录开始时间和结束时间。

Example 包含元数据的子分段

```
"subsegments": [{
```

```
  "id": "6f1605cd8a07cb70",
  "start_time": 1.480305974194E9,
  "end_time": 1.4803059742E9,
  "name": "Custom subsegment for UserModel.saveUser function",
  "metadata": {
    "debug": {
      "test": "Metadata string from UserModel.saveUser"
    }
  },
```

使用 X-Ray SDK for Python，将注释和元数据添加到分段

可以利用注释和元数据记录与请求、环境或应用程序相关的其他信息。可以将注释和元数据添加到 X-Ray 开发工具包创建的分段或您创建的自定义子分段。

注释是带字符串、数字或布尔值的键值对。系统会对注释编制索引，以便与筛选表达式一起使用。使用注释记录要用于对控制台中的跟踪进行分组的数据或在调用 `GetTraceSummaries` API 时使用的数据。

元数据是可以具有任何类型值的键-值对，包括对象和列表，但没有编制索引，无法与筛选条件表达式一起使用。使用元数据记录要存储在跟踪中但不需要用于搜索跟踪的其他数据。

除了注释和元数据之外，您还可以在分段上记录用户 ID 字符串。用户 IDs 被记录在区段的单独字段中，并编制索引以供搜索使用。

Sections

- 使用 X-Ray SDK for Python 记录注释
- 使用 X-Ray SDK for Python 记录元数据
- 使用适用于 Python 的 X-Ray SDK 录制用户 IDs

使用 X-Ray SDK for Python 记录注释

使用注释记录有关要为其编制索引以进行搜索的分段和子分段的信息。

注释要求

- 键 - X-Ray 注释的键最多可以包含 500 个字母数字字符。除了点或句点（.）之外，不能使用空格或符号
- 值 - X-Ray 注释的值最多可以包含 1,000 个 Unicode 字符。

- 注释的数量 - 每个跟踪最多可使用 50 条注释。

记录注释

1. 从 `xray_recorder` 获取对当前分段或子分段的引用。

```
from aws_xray_sdk.core import xray_recorder
...
document = xray_recorder.current_segment()
```

或

```
from aws_xray_sdk.core import xray_recorder
...
document = xray_recorder.current_subsegment()
```

2. 调用带有字符串键和布尔值、数字值或字符串值的 `put_annotation`。

```
document.put_annotation("mykey", "my value");
```

以下示例说明如何使用包含点和布尔值、数字值或字符串值的字符串键调用 `putAnnotation`。

```
document.putAnnotation("testkey.test", "my value");
```

此外，您还可以使用对 `put_annotation` 使用 `xray_recorder` 方法。此方法会记录当前子分段上的注释，如果未打开子分段，则记录分段上的注释。

```
xray_recorder.put_annotation("mykey", "my value");
```

开发工具包将注释以键-值对的形式记录在分段文档的 `annotations` 对象中。使用相同键调用两次 `put_annotation` 将覆盖同一分段或子分段上之前记录的值。

要查找具有带特定值的注释的跟踪，请在`annotation[`*`key`*`]`筛选表达式中使用 关键字。

使用 X-Ray SDK for Python 记录元数据

使用元数据记录有关您无需为其编制索引以进行搜索的分段或子分段的信息。元数据值可以是字符串、数字、布尔值或可序列化为 JSON 对象或数组的任何对象。

记录元数据

1. 从 xray_recorder 获取对当前分段或子分段的引用。

```
from aws_xray_sdk.core import xray_recorder
...
document = xray_recorder.current_segment()
```

或

```
from aws_xray_sdk.core import xray_recorder
...
document = xray_recorder.current_subsegment()
```

2. 使用字符串键、布尔值、数字、字符串或对象值以及字符串命名空间调用 put_metadata。

```
document.put_metadata("my key", "my value", "my namespace");
```

或

调用仅带有键和值的 put_metadata。

```
document.put_metadata("my key", "my value");
```

此外，您还可以使用对 put_metadata 使用 xray_recorder 方法。此方法会记录当前子分段上的元数据，如果未打开子分段，则记录分段上的元数据。

```
xray_recorder.put_metadata("my key", "my value");
```

如果您没有指定命名空间，则开发工具包将使用 default。使用相同键调用两次 put_metadata 将覆盖同一分段或子分段上之前记录的值。

使用适用于 Python 的 X-Ray SDK 录制用户 IDs

记录请求细分中的用户，以识别发送请求的用户。 IDs

要记录用户 IDs

1. 从 xray_recorder 获取对当前分段的引用。

```
from aws_xray_sdk.core import xray_recorder
...
document = xray_recorder.current_segment()
```

2. 使用发送请求的用户的字符串 ID 调用 `setUser`。

```
document.set_user("U12345");
```

您可以在控制器中调用 `set_user` 以便在应用程序开始处理请求后立即记录用户 ID。

要查找用户 ID 的跟踪，请在`user`筛选表达式中使用 关键字。

检测部署到无服务器环境的 Web 框架

适用于 Python 的 AWS X-Ray SDK 支持检测部署在无服务器应用程序中的网络框架。无服务器是云端原生架构，可让您将更多的运营职责切换到 AWS，帮助您提升灵活性和创新。

无服务器架构是一种软件应用程序模式，让您无需考虑服务器的问题即可构建和运行应用程序和服务。为您省去基础设施管理任务，例如，服务器或集群预配、修补、操作系统维护，以及容量预配。几乎可以为任何类型的应用程序或后端服务构建无服务器解决方案，即可为您处理好运行和缩放高可用性应用程序所需要的一切。

本教程向您展示了如何在部署到无服务器 AWS X-Ray 环境的 Web 框架（例如 Flask 或 Django）上自动进行检测。应用程序的 X-Ray 检测允许您查看所有下游调用，从 Amazon API Gateway 到您的 AWS Lambda 函数，以及您的应用程序发出的传出调用。

适用于 Python 的 X-Ray 开发工具包支持以下 Python 应用程序框架：

- FLASK 版本 0.8 或更高版本
- Django 版本 1.0 或更高版本

本教程开发了一款示例无服务器应用程序部署到 Lambda，由 API Gateway 调用。本教程使用 Zappa 自动将应用程序部署到 Lambda 并配置 API Gateway 端点。

先决条件

- Zappa
- Python - 版本 2.7 或 3.6。

- AWS CLI— 验证您的账户 AWS CLI 是否配置 AWS 区域 了以及您将在其中部署应用程序。
- Pip
- Virtualenv

步骤 1：创建 环境

在此步骤中，您将创建一个使用 `virtualenv` 托管应用程序的虚拟环境。

1. 使用 AWS CLI，为应用程序创建目录。然后切换到新目录。

```
mkdir serverless_application
cd serverless_application
```

2. 接下来，在新目录中创建一个虚拟环境。请使用以下命令激活虚拟环境。

```
# Create our virtual environment
virtualenv serverless_env

# Activate it
source serverless_env/bin/activate
```

3. 将 X-Ray、Flask、Zappa 和请求库安装到您的环境中。

```
# Install X-Ray, Flask, Zappa, and Requests into your environment
pip install aws-xray-sdk flask zappa requests
```

4. 将应用程序代码添加到 `serverless_application` 目录中。在这个例子中，我们可以构建 Flash 的 Hello World 示例。

   在 `serverless_application` 目录中创建名为 `my_app.py` 的文件。然后使用文本编辑器添加以下命令。此应用程序将检测 Requests 库，修补 Flask 应用程序的中间件，并打开端点 `'/'`。

```
# Import the X-Ray modules
from aws_xray_sdk.ext.flask.middleware import XRayMiddleware
from aws_xray_sdk.core import patcher, xray_recorder
from flask import Flask
import requests

# Patch the requests module to enable automatic instrumentation
patcher.patch(('requests',))
```

```
app = Flask(__name__)

# Configure the X-Ray recorder to generate segments with our service name
xray_recorder.configure(service='My First Serverless App')

# Instrument the Flask application
XRayMiddleware(app, xray_recorder)

@app.route('/')
def hello_world():
    resp = requests.get("https://aws.amazon.com")
    return 'Hello, World: %s' % resp.url
```

步骤 2：创建并部署一个 Zappa 环境

在此步骤中，您将使用 Zappa 自动配置 API Gateway 端点，然后部署到 Lambda。

1. 在 `serverless_application` 目录里初始化 Zappa。在本示例中，我们使用了默认设置，但是如果您有自定义首选项，Zappa 会显示配置说明。

```
zappa init
```

```
What do you want to call this environment (default 'dev'): dev
...
What do you want to call your bucket? (default 'zappa-*******'): zappa-*******
...
...
It looks like this is a Flask application.
What's the modular path to your app's function?
This will likely be something like 'your_module.app'.
We discovered: my_app.app
Where is your app's function? (default 'my_app.app'): my_app.app
...
Would you like to deploy this application globally? (default 'n') [y/n/
(p)rimary]: n
```

2. 启用 X-Ray。打开 `zappa_settings.json` 文件并验证其外观是否与示例相似。

```
{
    "dev": {
        "app_function": "my_app.app",
```

```
        "aws_region": "us-west-2",
        "profile_name": "default",
        "project_name": "serverless-exam",
        "runtime": "python2.7",
        "s3_bucket": "zappa-*********"
    }
}
```

3. 将 `"xray_tracing": true` 作为空目录添加到配置文件。

```
{
    "dev": {
        "app_function": "my_app.app",
        "aws_region": "us-west-2",
        "profile_name": "default",
        "project_name": "serverless-exam",
        "runtime": "python2.7",
        "s3_bucket": "zappa-*********",
        "xray_tracing": true
    }
}
```

4. 部署 应用程序。这会自动配置 API Gateway 端点并将您的代码上传到 Lambda。

```
zappa deploy
```

```
...
Deploying API Gateway..
Deployment complete!: https://**********.execute-api.us-west-2.amazonaws.com/dev
```

步骤 3：为 API Gateway 启用 X-Ray 跟踪

在此步骤中，您将与 API Gateway 控制台进行交互以启用 X-Ray 跟踪。

1. 登录 AWS Management Console 并打开 API Gateway 控制台，网址为https://console.aws.amazon.com/apigateway/。
2. 找到新生成的 API。它应该类似于 `serverless-exam-dev`。
3. 选择阶段。
4. 选择部署阶段的名称。默认为 `dev`。

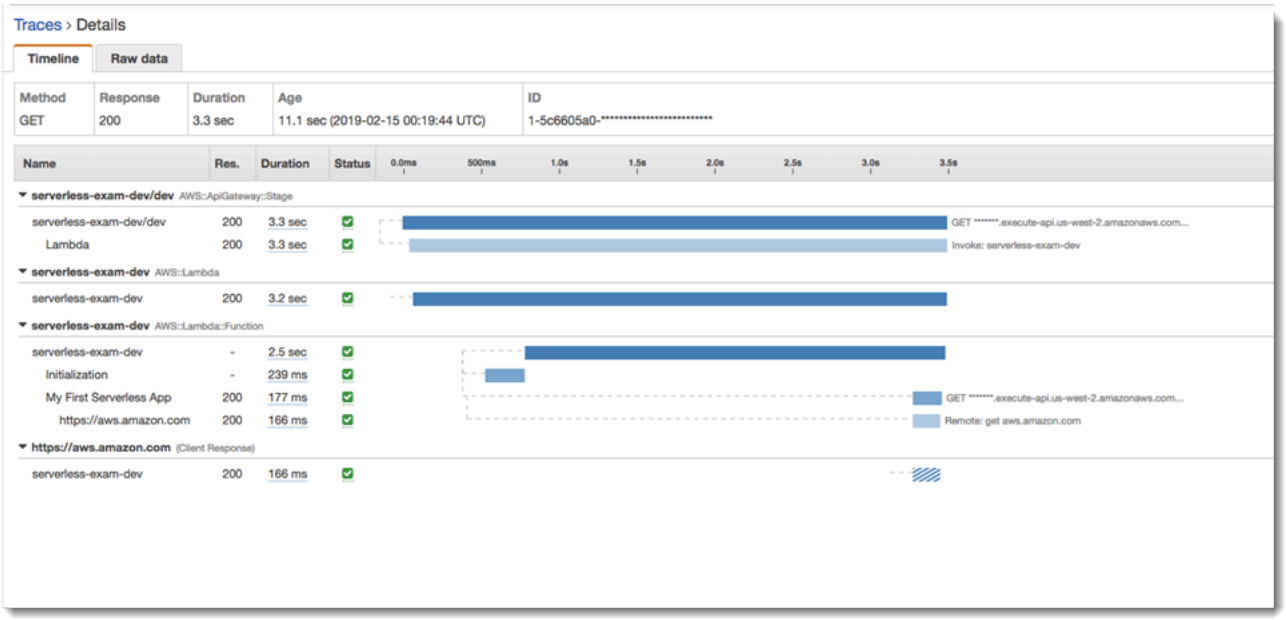
5. 在日志/跟踪选项卡上，选中启用 X-Ray 跟踪复选框。
6. 选择保存更改。
7. 在浏览器中访问端点。如果您使用了示例 `Hello World` 应用程序，它应显示以下内容。

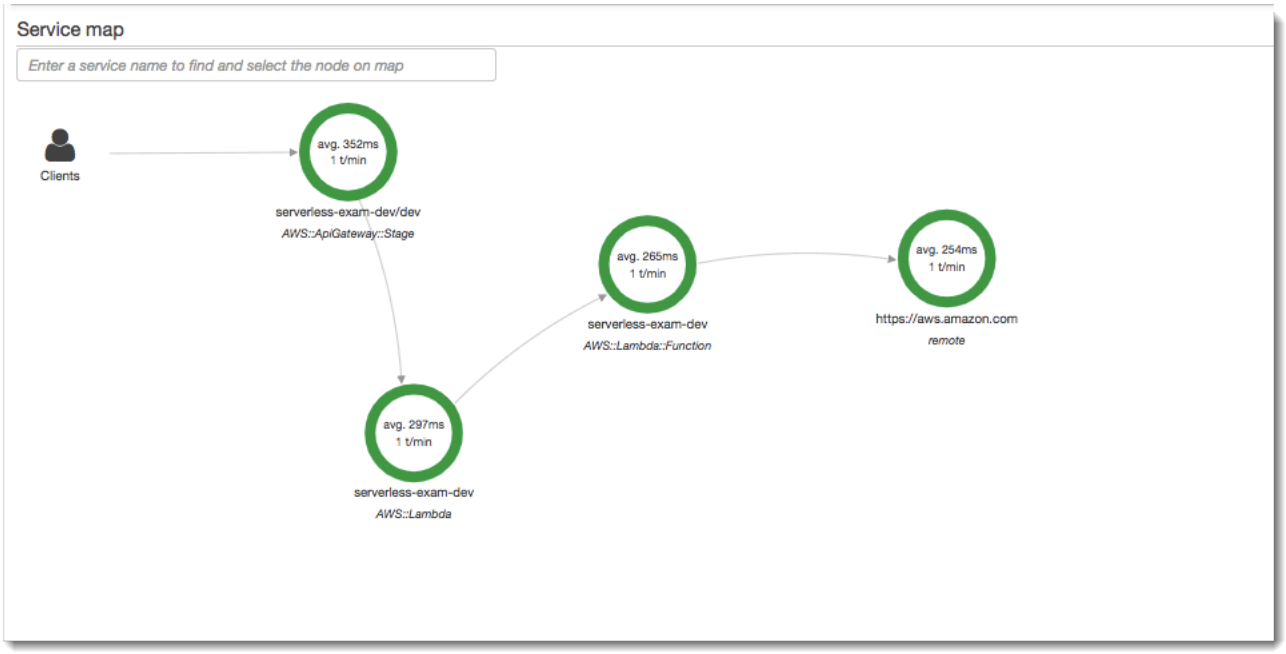
```
"Hello, World: https://aws.amazon.com/"
```

步骤 4：查看创建的跟踪

在此步骤中，您将与 X-Ray 控制台交互，以查看示例应用程序所创建的跟踪。有关跟踪分析的更详细演练，请参阅查看服务映射。

1. 登录 AWS Management Console 并在https://console.aws.amazon.com/xray/家中打开 X-Ray 控制台。
2. 查看 API Gateway、Lambda 函数和 Lambda 容器生成的分段。
3. 在 Lambda 函数分段下，查看名为 `My First Serverless App` 的子分段。紧随其后的是名为 `https://aws.amazon.com` 的第二个子段。
4. 在初始化期间，Lambda 可能还会生成名为 `initialization` 的第三个子分段。





第 5 步：清理

始终终止不再使用的资源，以避免意外的成本累积。正如本教程所演示的那样，Zappa 此类工具可以简化无服务器部署。

要从 Lambda、API Gateway 和 Amazon S3中删除应用程序，请通过使用 AWS CLI从项目目录中运行以下命令。

```
zappa undeploy dev
```

后续步骤

通过添加 AWS 客户端并使用 X-Ray 对其进行检测，为您的应用程序添加更多功能。若要详细了解无服务器计算选项，请前往 AWS上的无服务器。

使用 .NET

有两种方法可用于检测 .NET 应用程序，以将跟踪数据发送到 X-Ray：

- AWS 适用于 OpenTelemetry .NET 的 Distro — 提供一组开源库的 AWS 发行版，用于通过 Distro for Collect AWS o r Collector 向多个 AWS 监控解决方案（包括亚马逊和亚马逊 OpenSearch 服务）发送相关的指标和跟踪。 CloudWatch AWS X-Ray OpenTelemetry
- AWS X-Ray 适用于.NET 的 SDK — 一组库，用于通过 X-Ray 守护程序生成跟踪并将其发送到 X-Ray。

有关更多信息，请参阅 在 AWS Distro for 和 X-Ray OpenTelemetry 之间进行选择 SDKs。

AWS .NET 发行 OpenTelemetry 版

使用适用于 OpenTelemetry .NET 的 AWS Distro，您可以对应用程序进行一次检测，然后将相关的指标和跟踪发送到多个 AWS 监控解决方案 CloudWatch AWS X-Ray，包括亚马逊和亚马逊 OpenSearch 服务。将 X-Ray 与 AWS Distro 配合使用 OpenTelemetry 需要两个组件：启用 OpenTelemetry SDK 以与 X-Ray 配合使用，以及启用 Collecto OpenTelemetry r 的AWS Distro 以与 X-Ray 配合使用。

要开始使用，请参阅 OpenTelemetry .NET AWS 发行版文档。

有关将 Distro 用于 with AWS X-Ray 和 other 的 OpenTelemetry 更多信息 AWS 服务，请参阅 AWS Distro for OpenTelemetry 或 AWS Distro for AWS D ocuments。 OpenTelemetry

有关语言支持和用法的更多信息，请参阅上的 O AWS bservability。 GitHub

AWS X-Ray 适用于.NET 的 SDK

适用于.NET 的 X-Ray SDK 是一个用于检测 C# .NET Web 应用程序、.NET Core Web 应用程序和.NET 核心函数的库。 AWS Lambda它提供用于生成跟踪数据并将其发送到 X-Ray 进程守护程序的类及方法。这包括有关应用程序处理的传入请求的信息，以及应用程序对下游 AWS 服务、HTTP Web APIs 和 SQL 数据库的调用的信息。

> **Note**
>
> X-Ray SDK for .NET 是一个开源项目。你可以关注该项目并在 github 上 GitHub提交议题和拉取请求。 com/aws/aws-xray-sdk-dotnet

对于 Web 应用程序，首先通过添加消息处理程序到 Web 配置来跟踪传入请求。消息处理程序为每个被跟踪的请求创建一个分段并在发送响应时完成该分段。当分段打开时，您可以使用开发工具包客户端的方法将信息添加到分段，并创建子分段以跟踪下游调用。开发工具包还会自动记录在分段打开时应用程序引发的异常。

对于由经过检测的应用程序或服务调用的 Lambda 函数，Lambda 会读取跟踪标头并自动跟踪采样的请求。对于其他函数，您可以将 Lambda 配置为采样和跟踪传入请求。无论哪种情况，Lambda 都会创建分段并将其提供给 X-Ray 开发工具包。

> **Note**
>
> 在 Lambda 上，X-Ray 开发工具包是可选的。如果您不在函数中使用它，您的服务映射仍将包含一个用于 Lambda 服务的节点，以及每个 Lambda 函数的节点。可通过添加该开发工具包检测函数代码，将子分段添加到 Lambda 记录的函数分段。请参阅AWS Lambda 和 AWS X-Ray了解更多信息。

接下来，使用 X-Ray SDK for .NET 检测 适用于 .NET 的 AWS SDK 客户端。每当您使用已检测的客户端调用下游 AWS 服务 或资源时，SDK 都会在子分段中记录有关该调用的信息。 AWS 服务和您在服务中访问的资源在跟踪图上显示为下游节点，以帮助您识别各个连接上的错误和限制问题。

适用于.NET 的 X-Ray SDK 还为对 HTTP Web APIs 和 SQL 数据库的下游调用提供了工具。`GetResponseTraced` 的 `System.Net.HttpWebRequest` 扩展方法跟踪传出 HTTP 调用。您可以使用 X-Ray SDK for .NET 的 `SqlCommand` 版本来检测 SQL 查询。

在开始使用 SDK 后，通过配置记录器和消息处理程序来自定义其行为。您可以添加插件来记录有关应用程序上运行的计算资源的数据，通过定义采样规则来自定义采样行为，设置日志级别以在应用程序日志中查看来自开发工具包的更多或更少的信息。

记录有关请求以及应用程序在注释和元数据中所做的工作的其他信息。注释是简单的键值对，已为这些键值对编制索引以用于筛选条件表达式，以便您能够搜索包含特定数据的跟踪。元数据条目的限制性较低，并且可以记录整个对象和数组 - 可序列化为 JSON 的任何项目。

> **注释和元数据**
>
> 注释和元数据是您使用 X-Ray 开发工具包添加到分段的任意文本。系统会对注释编制索引，以便与筛选表达式一起使用。元数据未编制索引，但可以使用 X-Ray 控制台或 API 在原始分段中查看。您授予 X-Ray 读取权限的任何人都可以查看这些数据。

当代码中具有大量检测的客户端时，一个请求分段可包含大量子分段，检测的客户端发起的每个调用均对应一个子分段。您可以通过将客户端调用包含在自定义子分段中来整理子分段并为其分组。您可以为整个函数或任何代码部分创建自定义子分段，并记录子分段的元数据和注释，而不是编写父分段的所有内容。

有关 SDK 的类和方法的参考文档，请参阅以下内容：

- AWS X-Ray 适用于.NET 的 SDK API 参考
- AWS X-Ray 适用于.NET 的 SDK Core API 参考

同一个程序包同时支持 .NET 和 .NET Core，但使用的类不同。本章中的示例与 .NET API 参考相关，除非该类特定于 .NET Core。

要求

适用于.NET 的 X-Ray SDK 需要.NET 框架 4.5 或更高版本以及 适用于 .NET 的 AWS SDK。

对于 .NET Core 应用程序和函数，SDK 需要 .NET Core 2.0 或更高版本。

将 X-Ray SDK for .NET 添加到应用程序

用于 NuGet 将适用于.NET 的 X-Ray SDK 添加到您的应用程序中。

在 Visual Studio 中使用 NuGet 包管理器安装适用于.NET 的 X-Ray SDK

1. 选择 “工具”、“P NuGet ackage Manager”、“管理解决方案 NuGet 包”。
2. 搜索AWSXRay录音机。
3. 依次选择此程序包和安装。

依赖关系管理

可从 Nuget 获得 X-Ray SDK for .NET。使用程序包管理器安装 SDK ：

```
Install-Package AWSXRayRecorder -Version 2.10.1
```

AWSXRayRecorder v2.10.1 NuGet 程序包具有以下依赖项：

.NET Framework 4.5

```
AWSXRayRecorder (2.10.1)
|
|-- AWSXRayRecorder.Core (>= 2.10.1)
|   |-- AWSSDK.Core (>= 3.3.25.1)
|
|-- AWSXRayRecorder.Handlers.AspNet (>= 2.7.3)
|   |-- AWSXRayRecorder.Core (>= 2.10.1)
|
|-- AWSXRayRecorder.Handlers.AwsSdk (>= 2.8.3)
|   |-- AWSXRayRecorder.Core (>= 2.10.1)
|
|-- AWSXRayRecorder.Handlers.EntityFramework (>= 1.1.1)
|   |-- AWSXRayRecorder.Core (>= 2.10.1)
|   |-- EntityFramework (>= 6.2.0)
|
|-- AWSXRayRecorder.Handlers.SqlServer (>= 2.7.3)
|   |-- AWSXRayRecorder.Core (>= 2.10.1)
|
|-- AWSXRayRecorder.Handlers.System.Net (>= 2.7.3)
    |-- AWSXRayRecorder.Core (>= 2.10.1)
```

.NET Framework 2.0

```
AWSXRayRecorder (2.10.1)
|
|-- AWSXRayRecorder.Core (>= 2.10.1)
|   |-- AWSSDK.Core (>= 3.3.25.1)
|   |-- Microsoft.AspNetCore.Http (>= 2.0.0)
|   |-- Microsoft.Extensions.Configuration (>= 2.0.0)
|   |-- System.Net.Http (>= 4.3.4)
|
|-- AWSXRayRecorder.Handlers.AspNetCore (>= 2.7.3)
|   |-- AWSXRayRecorder.Core (>= 2.10.1)
```

```
|   |-- Microsoft.AspNetCore.Http.Extensions (>= 2.0.0)
|   |-- Microsoft.AspNetCore.Mvc.Abstractions (>= 2.0.0)
|
|-- AWSXRayRecorder.Handlers.AwsSdk (>= 2.8.3)
|   |-- AWSXRayRecorder.Core (>= 2.10.1)
|
|-- AWSXRayRecorder.Handlers.EntityFramework (>= 1.1.1)
|   |-- AWSXRayRecorder.Core (>= 2.10.1)
|   |-- Microsoft.EntityFrameworkCore.Relational (>= 3.1.0)
|
|-- AWSXRayRecorder.Handlers.SqlServer (>= 2.7.3)
|   |-- AWSXRayRecorder.Core (>= 2.10.1)
|   |-- System.Data.SqlClient (>= 4.4.0)
|
|-- AWSXRayRecorder.Handlers.System.Net (>= 2.7.3)
    |-- AWSXRayRecorder.Core (>= 2.10.1)
```

有关依赖项管理的详细信息，请参阅 Microsoft 关于 Nuget 依赖项和 Nuget 依赖项解析的文档。

配置适用于 .NET 的 X-Ray 开发工具包

您可以配置带有插件的适用于 .NET 的 X-Ray 开发工具包 以包括应用程序在其上运行的服务的相关信息，修改默认采样行为，或者添加应用于特定路径请求的采样规则。

对于 .NET Web 应用程序，请将密钥添加到 `appSettings` 文件的 `Web.config` 部分。

Example Web.config

```
<configuration>
  <appSettings>
    <add key="AWSXRayPlugins" value="EC2Plugin"/>
    <add key="SamplingRuleManifest" value="sampling-rules.json"/>
  </appSettings>
</configuration>
```

对于 .NET Core，请使用名为 `appsettings.json` 的顶层密钥创建名为 `XRay` 的文件。

Example .NET appsettings.json

```
{
  "XRay": {
    "AWSXRayPlugins": "EC2Plugin",
```

```
    "SamplingRuleManifest": "sampling-rules.json"
  }
}
```

然后，在应用程序代码中，生成配置对象并将其用于初始化 X-Ray 记录器。在初始化记录器之前执行此操作。

Example .NET Core Program.cs 记录器配置

```
using Amazon.XRay.Recorder.Core;
...
AWSXRayRecorder.InitializeInstance(configuration);
```

如果您正在检测 .NET Core Web 应用程序，则在配置消息处理程序时，还可以将配置对象传递到 UseXRay 方法。对于 Lambda 函数，使用上面所示的 `InitializeInstance` 方法。

有关 .NET Core 配置 API 的更多信息，请参阅 docs.microsoft.com 上的配置 ASP.NET Core 应用程序。

Sections

- 插件
- 采样规则
- 日志记录 (.NET)
- 日志记录 (.NET Core)
- 环境变量

插件

使用插件可添加有关托管您应用程序的服务的数据。

插件

- Amazon EC2 — `EC2Plugin` 添加实例 ID、可用区和 CloudWatch 日志组。
- Elastic Beanstalk - `ElasticBeanstalkPlugin` 添加环境名称、版本标签和部署 ID。
- Amazon ECS — `ECSPlugin` 添加容器 ID。

要使用插件，请通过添加 `AWSXRayPlugins` 设置配置适用于 .NET 的 X-Ray 开发工具包客户端。如果多个插件应用到您的应用程序，请在同一个设置中指定所有这些设置，以逗号分隔。

Example Web.config - 插件

```
<configuration>
  <appSettings>
    <add key="AWSXRayPlugins" value="EC2Plugin,ElasticBeanstalkPlugin"/>
  </appSettings>
</configuration>
```

Example .NET Core appsettings.json 插件

```
{
  "XRay": {
    "AWSXRayPlugins": "EC2Plugin,ElasticBeanstalkPlugin"
  }
}
```

采样规则

该 SDK 使用您在 X-Ray 控制台中定义的采样规则来确定要记录的请求。默认规则跟踪每秒的第一个请求，以及所有将跟踪发送到 X-Ray 的服务的任何其他请求的百分之五。在 X-Ray 控制台中创建其他规则以自定义为每个应用程序记录的数据量。

该 SDK 按照定义的顺序应用自定义规则。如果请求与多个自定义规则匹配，则 SDK 仅应用第一条规则。

> **Note**
>
> 如果 SDK 无法访问 X-Ray 来获取采样规则，它将恢复为默认的本地规则，即每秒第一个请求以及每个主机所有其他请求的百分之五。如果主机无权调用采样，或者无法连接到 X-Ray 守护程序 APIs，后者充当 SDK 发出的 API 调用的 TCP 代理，则可能会发生这种情况。

您还可以将 SDK 配置为从 JSON 文档加载采样规则。在 X-Ray 采样不可用的情况下，SDK 可以使用本地规则作为备份，也可以只使用本地规则。

Example sampling-rules.json

```
{
  "version": 2,
  "rules": [
```

```
    {
      "description": "Player moves.",
      "host": "*",
      "http_method": "*",
      "url_path": "/api/move/*",
      "fixed_target": 0,
      "rate": 0.05
    }
  ],
  "default": {
    "fixed_target": 1,
    "rate": 0.1
  }
}
```

此示例定义了一个自定义规则和一个默认规则。自定义规则采用百分之五的采样率，对于 `/api/move/` 之下的路径要跟踪的请求数量不设下限。默认规则中每秒的第一个请求以及其他请求的百分之十。

在本地定义规则的缺点是，固定目标由记录器的每个实例独立应用而不是由 X-Ray 服务管理。随着您部署更多主机，固定速率会成倍增加，这使得控制记录的数据量变得更加困难。

开启后 AWS Lambda，您无法修改采样率。如果您的函数由检测服务调用，Lambda 将记录生成由该服务采样的请求的调用。如果启用了主动跟踪且不存在任何跟踪标头，则 Lambda 会做出采样决定。

要配置备份规则，请指示从具有适用于 .NET 的 X-Ray 开发工具包 `SamplingRuleManifest` 设置的文件加载采样规则。

Example .NET Web.config - 采样规则

```
<configuration>
  <appSettings>
    <add key="SamplingRuleManifest" value="sampling-rules.json"/>
  </appSettings>
</configuration>
```

Example .NET Core appsettings.json 采样规则

```
{
  "XRay": {
    "SamplingRuleManifest": "sampling-rules.json"
```

```
  }
}
```

要仅使用本地规则，请使用 LocalizedSamplingStrategy 构建记录器。如果您配置了备份规则，请删除该配置。

Example .NET global.asax - 本地采样规则

```
var recorder = new AWSXRayRecorderBuilder().WithSamplingStrategy(new
 LocalizedSamplingStrategy("samplingrules.json")).Build();
AWSXRayRecorder.InitializeInstance(recorder: recorder);
```

Example .NET Core Program.cs - 本地采样规则

```
var recorder = new AWSXRayRecorderBuilder().WithSamplingStrategy(new
 LocalizedSamplingStrategy("sampling-rules.json")).Build();
AWSXRayRecorder.InitializeInstance(configuration,recorder);
```

日志记录 (.NET)

适用于 .NET 的 X-Ray 开发工具包使用与 适用于 .NET 的 AWS SDK 相同的日志记录机制。如果您已经将应用程序配置为记录 适用于 .NET 的 AWS SDK 输出，则同样的配置也适用于适用于.NET 的 X-Ray SDK 的输出。

要配置日志记录，请将名为 aws 的配置部分添加到您的 App.config 文件或 Web.config 文件。

Example Web.config - 日志记录

```
...
<configuration>
  <configSections>
    <section name="aws" type="Amazon.AWSSection, AWSSDK.Core"/>
  </configSections>
  <aws>
    <logging logTo="Log4Net"/>
  </aws>
</configuration>
```

有关更多信息，请参阅 适用于 .NET 的 AWS SDK 开发人员指南 中的配置您的适用于 .NET 的 AWS SDK 应用程序。

日志记录 (.NET Core)

适用于 .NET 的 X-Ray 开发工具包使用与 适用于 .NET 的 AWS SDK 相同的日志记录选项。要为.NET Core 应用程序配置日志记录，请将日志选项传递给 `AWSXRayRecorder.RegisterLogger` 方法。

例如，要使用 log4net，请创建定义记录器的配置文件、输出格式和文件位置。

Example .NET Core log4net.config

```
<?xml version="1.0" encoding="utf-8" ?>
<log4net>
  <appender name="FileAppender" type="log4net.Appender.FileAppender,log4net">
    <file value="c:\logs\sdk-log.txt" />
    <layout type="log4net.Layout.PatternLayout">
      <conversionPattern value="%date [%thread] %level %logger - %message%newline" />
    </layout>
  </appender>
  <logger name="Amazon">
    <level value="DEBUG" />
    <appender-ref ref="FileAppender" />
  </logger>
</log4net>
```

然后，创建记录器并在程序代码中应用配置。

Example .NET Core Program.cs - 日志记录

```
using log4net;
using Amazon.XRay.Recorder.Core;

class Program
{
  private static ILog log;
  static Program()
  {
    var logRepository = LogManager.GetRepository(Assembly.GetEntryAssembly());
    XmlConfigurator.Configure(logRepository, new FileInfo("log4net.config"));
    log = LogManager.GetLogger(typeof(Program));
    AWSXRayRecorder.RegisterLogger(LoggingOptions.Log4Net);
  }
  static void Main(string[] args)
  {
  ...
```

```
  }
}
```

有关配置 log4net 的更多信息，请参阅 logging.apache.org 上的配置。

环境变量

您可以使用环境变量配置适用于.NET 的 X-Ray 开发工具包。SDK 支持以下变量。

- `AWS_XRAY_TRACING_NAME` - 设置开发工具包用于进行分段的服务名称。覆盖您根据 servlet 筛选器的分段命名策略设置的服务名称。
- `AWS_XRAY_DAEMON_ADDRESS` - 设置 X-Ray 进程守护程序侦听器的主机和端口。默认情况下，SDK 使用用于跟踪数据（UDP）和采样（TCP）的 `127.0.0.1:2000`。如果您已将进程守护程序配置为侦听不同端口或者进程守护程序在另一台主机上运行，则使用此变量。

  格式

  - 同一个端口 — *`address`*`:`*`port`*
  - 不同的端口 — `tcp:`*`address`*`:`*`port`* `udp:`*`address`*`:`*`port`*
- `AWS_XRAY_CONTEXT_MISSING` - 设置为 `RUNTIME_ERROR` 在您的已检测代码尝试在分段未打开的情况下记录数据时引发异常。

  有效值

  - `RUNTIME_ERROR`— 引发运行时异常。
  - `LOG_ERROR`— 记录错误并继续（默认）。
  - `IGNORE_ERROR`— 忽略错误并继续。

  对于在未打开任何请求时运行的启动代码或者会生成新线程的代码，如果您尝试在其中使用检测过的客户端，则可能发生与缺失分段或子分段相关的错误。

使用适用于 .NET 的 X-Ray 开发工具包检测传入 HTTP 请求

您可以使用 X-Ray SDK 来跟踪您的应用程序在亚马逊或亚马逊 EC2 ECS 中的 EC2 实例上提供的传入 HTTP 请求。 AWS Elastic Beanstalk

使用消息处理程序检测传入 HTTP 请求。当您将 X-Ray 消息处理程序添加到应用程序时，适用于 .NET 的 X-Ray 开发工具包将为每个采样请求创建分段。此分段包括 HTTP 请求的计时、方法和处置。其他检测会在此分段上创建子分段。

> **Note**
>
> 对于 AWS Lambda 函数，Lambda 会为每个采样请求创建一个分段。请参阅AWS Lambda 和 AWS X-Ray了解更多信息。

每个分段都有一个名称，用于在服务映射中标识您的应用程序。可以静态命名分段，也可以将 SDK 配置为根据传入请求中的主机标头对其进行动态命名。动态命名允许根据请求中的域名对跟踪进行分组，并且在名称不匹配预期模式时（例如，如果主机标头是伪造的）应用默认名称。

> **转发的请求**
>
> 如果负载均衡器或其他中间将请求转发到您的应用程序，X-Ray 会提取请求 `X-Forwarded-For` 标头中的客户端 IP 而非 IP 数据包中的源 IP。由于转发的请求记录的客户端 IP 可以伪造，因此不应信任。

信息处理程序使用包含以下信息的 `http` 数据块为每个传入请求创建一个分段：

- HTTP 方法 - GET、POST、PUT、DELETE 等。
- 客户端地址 - 发送请求的客户端的 IP 地址。
- 响应代码 - 已完成请求的 HTTP 响应代码。
- 时间 - 开始时间（收到请求时）和结束时间（发送响应时）。
- 用户代理 - 请求中的 `user-agent`。
- 内容长度 - 响应中的 `content-length`。

Sections

检测传入请求 (.NET)

要检测由您的应用程序所服务的请求，请在 `global.asax` 文件的 `Init` 方法中调用 `RegisterXRay`。

Example global.asax - 消息处理程序

```
using System.Web.Http;
using Amazon.XRay.Recorder.Handlers.AspNet;

namespace SampleEBWebApplication
{
  public class MvcApplication : System.Web.HttpApplication
  {
    public override void Init()
    {
      base.Init();
      AWSXRayASPNET.RegisterXRay(this, "MyApp");
    }
  }
}
```

检测传入请求 (.NET Core)

若要检测您的应用程序处理的请求，请在启动类的 Configure 方法中的任何其他中间件之前调用 UseXRay 方法。因为理想情况下，X-Ray 应该是第一个处理请求的中间件，以及最后一个处理管道中响应的中间件。

> **Note**
>
> 对于.NET Core 2.0，如果应用程序中有 UseExceptionHandler 方法，请确保在 UseExceptionHandler 方法之后调用 UseXRay 以确保记录下异常。

Example Startup.cs

<caption>.NET Core 2.1 and above</caption>

```
using Microsoft.AspNetCore.Builder;

public void Configure(IApplicationBuilder app, IHostingEnvironment env)
  {
    app.UseXRay("MyApp");
    // additional middleware
    ...
  }
```

<caption>.NET Core 2.0</caption>

```
using Microsoft.AspNetCore.Builder;

public void Configure(IApplicationBuilder app, IHostingEnvironment env)
  {
    app.UseExceptionHandler("/Error");
    app.UseXRay("MyApp");
    // additional middleware
    ...
  }
```

UseXRay 方法还会获取配置对象作为第二个参数。

```
app.UseXRay("MyApp", configuration);
```

配置分段命名策略

AWS X-Ray 使用服务名称来标识您的应用程序，并将其与您的应用程序使用的其他应用程序、数据库 APIs、外部数据库和 AWS 资源区分开来。当 X-Ray SDK 为传入请求生成分段时，会将应用程序的服务名称记录在分段的名称字段中。

X-Ray SDK 可以用在 HTTP 请求标头中的 hostname 来命名分段。不过，此标头可以伪造，会导致服务地图中出现意料之外的节点。为防止 SDK 由于包含伪造的主机标头的请求而错误地命名分段，必须为传入请求指定一个默认名称。

如果应用程序为多个域的请求提供服务，则可以将 SDK 配置为使用动态命名策略以在分段名称中反映出这一点。动态命名策略允许 SDK 将主机名用于符合预期模式的请求，并将默认名称应用于不符合预期模式的请求。

例如，可能有一款应用程序为发送到三个子域的请求提供服务，分别为 `www.example.com`、`api.example.com` 和 `static.example.com`。可以使用格式 `*.example.com` 的动态命名策略以识别包含不同名称的子域的分段，服务地图上因此会显示三个服务节点。如果应用程序收到包含与该格式不匹配的 hostname 的请求，您将会在服务地图上看到第四个节点，以及您指定的回退名称。

要对所有请求分段使用同一名称，可在初始化消息处理程序时指定应用程序名称，如上一部分中所示。这与创建 `FixedSegmentNamingStrategy` 并将它传递给 `RegisterXRay` 方法的效果相同。

```
AWSXRayASPNET.RegisterXRay(this, new FixedSegmentNamingStrategy("MyApp"));
```

> **Note**
>
> 您可以使用 AWS_XRAY_TRACING_NAME 环境变量覆盖您在代码中定义的默认服务名称。

动态命名策略定义一个主机名应匹配的模式和一个在 HTTP 请求中的主机名与该模式不匹配时要使用的默认名称。要动态命名分段，请创建 DynamicSegmentNamingStrategy 并将它传递给 RegisterXRay 方法。

```
AWSXRayASPNET.RegisterXRay(this, new DynamicSegmentNamingStrategy("MyApp",
 "*.example.com"));
```

使用适用于.NET 的 X-Ray SD AWS K 追踪 SDK 调用

当您的应用程序调用 AWS 服务 以存储数据、写入队列或发送通知时，X-Ray SDK for .NET 会按子分段跟踪下游的调用。在这些服务（例如，Amazon S3 存储桶或 Amazon SQS 队列）中追踪的资源 AWS 服务 和访问的资源在 X-Ray 控制台的跟踪地图上显示为下游节点。

在创建 适用于 .NET 的 AWS SDK 客户RegisterXRayForAllServices之前，您可以通过致电来检测所有客户。

Example SampleController.cs-DynamoDB 客户端工具

```
using Amazon;
using Amazon.Util;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DocumentModel;
using Amazon.XRay.Recorder.Core;
using Amazon.XRay.Recorder.Handlers.AwsSdk;

namespace SampleEBWebApplication.Controllers
{
  public class SampleController : ApiController
  {
    AWSSDKHandler.RegisterXRayForAllServices();
    private static readonly Lazy<AmazonDynamoDBClient> LazyDdbClient = new
 Lazy<AmazonDynamoDBClient>(() =>
    {
      var client = new AmazonDynamoDBClient(EC2InstanceMetadata.Region ??
 RegionEndpoint.USEast1);
      return client;
```

```
    });
```

要检测一些服务的客户端而不包括另一些服务的客户端，请调用 RegisterXRay 而不是 RegisterXRayForAllServices。使用服务客户端接口的名称替换突出显示的文本。

```
AWSSDKHandler.RegisterXRay<IAmazonDynamoDB>()
```

对于所有服务，都可以在 X-Ray 控制台中看到调用的 API 的名称。X-Ray 开发工具包会为一部分服务将信息添加到分段，从而在服务地图中提供更高的粒度。

例如，当使用经过检测的 DynamoDB 客户端发出调用时，对于针对表的调用，开发工具包会将表名称添加到分段中。在控制台中，每个表在服务地图中显示为一个独立的节点，以及没有表作为目标的调用的一般 DynamoDB 节点。

Example 对 DynamoDB 进行调用以保存项目的子分段

```
{
  "id": "24756640c0d0978a",
  "start_time": 1.480305974194E9,
  "end_time": 1.4803059742E9,
  "name": "DynamoDB",
  "namespace": "aws",
  "http": {
    "response": {
      "content_length": 60,
      "status": 200
    }
  },
  "aws": {
    "table_name": "scorekeep-user",
    "operation": "UpdateItem",
    "request_id": "UBQNSO5AEM8T4FDA4RQDEB94OVTDRVV4K4HIRGVJF66Q9ASUAAJG",
  }
}
```

在您访问指定的资源时，对以下服务的调用会在服务地图中创建额外的节点。没有定向到特定资源的调用，为服务创建了通用节点。

- Amazon DynamoDB - 表名称
- Amazon Simple Storage Service - 存储桶和键名称

- Amazon Simple Queue Service - 队列名称

使用适用于 .NET 的 X-Ray 开发工具包跟踪对下游 HTTP Web 服务的调用

当您的应用程序调用微服务或公共 HTTP 时 APIs，您可以使用适用于.NET 的 X-Ray SDK 的`GetResponseTraced`扩展方法`System.Net.HttpWebRequest`来检测这些调用，并将该 API 作为下游服务添加到服务图中。

Example HttpWebRequest

```
using System.Net;
using Amazon.XRay.Recorder.Core;
using Amazon.XRay.Recorder.Handlers.System.Net;

private void MakeHttpRequest()
{
  HttpWebRequest request = (HttpWebRequest)WebRequest.Create("http://names.example.com/
api");
  request.GetResponseTraced();
}
```

对于异步调用，请使用 `GetAsyncResponseTraced`。

```
request.GetAsyncResponseTraced();
```

如果您使用 `system.net.http.httpclient`，请使用 `HttpClientXRayTracingHandler` 委托处理程序来记录调用。

Example HttpClient

```
using System.Net.Http;
using Amazon.XRay.Recorder.Core;
using Amazon.XRay.Recorder.Handlers.System.Net;

private void MakeHttpRequest()
{
  var httpClient = new HttpClient(new HttpClientXRayTracingHandler(new
 HttpClientHandler()));
  httpClient.GetAsync(URL);
}
```

在您检测对下游 Web API 的调用时，适用于 .NET 的 X-Ray 开发工具包会使用有关 HTTP 请求和响应的信息记录子分段。X-Ray 使用子分段为 API 生成推断分段。

Example 下游 HTTP 调用的子分段

```
{
  "id": "004f72be19cddc2a",
  "start_time": 1484786387.131,
  "end_time": 1484786387.501,
  "name": "names.example.com",
  "namespace": "remote",
  "http": {
    "request": {
      "method": "GET",
      "url": "https://names.example.com/"
    },
    "response": {
      "content_length": -1,
      "status": 200
    }
  }
}
```

Example 下游 HTTP 调用的推断分段

```
{
  "id": "168416dc2ea97781",
  "name": "names.example.com",
  "trace_id": "1-62be1272-1b71c4274f39f122afa64eab",
  "start_time": 1484786387.131,
  "end_time": 1484786387.501,
  "parent_id": "004f72be19cddc2a",
  "http": {
    "request": {
      "method": "GET",
      "url": "https://names.example.com/"
    },
    "response": {
      "content_length": -1,
      "status": 200
    }
  },
  "inferred": true
```

```
}
```

使用适用于 .NET 的 X-Ray 开发工具包跟踪 SQL 查询

适用于 .NET 的 X-Ray 开发工具包为 `System.Data.SqlClient.SqlCommand` 提供了名为 `TraceableSqlCommand` 的包装程序类，您可以用来代替 `SqlCommand`。您可以使用 `TraceableSqlCommand` 类初始化 SQL 命令。

使用同步和异步方法跟踪 SQL 查询

以下示例显示如何使用 `TraceableSqlCommand` 来同步和异步自动跟踪 SQL Server 查询。

Example **`Controller.cs`** - SQL 客户端检测（异步）

```
using Amazon;
using Amazon.Util;
using Amazon.XRay.Recorder.Core;
using Amazon.XRay.Recorder.Handlers.SqlServer;

private void QuerySql(int id)
{
  var connectionString = ConfigurationManager.AppSettings["RDS_CONNECTION_STRING"];
  using (var sqlConnection = new SqlConnection(connectionString))
  using (var sqlCommand = new TraceableSqlCommand("SELECT " + id, sqlConnection))
  {
    sqlCommand.Connection.Open();
    sqlCommand.ExecuteNonQuery();
  }
}
```

您可以使用 `ExecuteReaderAsync` 方法异步执行查询。

Example **`Controller.cs`** - SQL 客户端检测（异步）

```
using Amazon;
using Amazon.Util;
using Amazon.XRay.Recorder.Core;
using Amazon.XRay.Recorder.Handlers.SqlServer;
private void QuerySql(int id)
{
  var connectionString = ConfigurationManager.AppSettings["RDS_CONNECTION_STRING"];
```

```
  using (var sqlConnection = new SqlConnection(connectionString))
  using (var sqlCommand = new TraceableSqlCommand("SELECT " + id, sqlConnection))
  {
    await sqlCommand.ExecuteReaderAsync();
  }
}
```

收集对 SQL Server 执行的 SQL 查询

您可以启用 SqlCommand.CommandText 的捕获作为 SQL 查询创建的子分段的一部分。SqlCommand.CommandText 显示为子分段 JSON 中的字段 sanitized_query。默认情况下，出于安全考虑，此功能处于禁用状态。

> **Note**
>
> 如果您在 SQL 查询中以明文形式包含敏感信息，请不要启用收集功能。

可以通过下列两种方式启用 SQL 查询：

- 在应用程序全局配置中将 CollectSqlQueries 属性设置为 true。
- 将 TraceableSqlCommand 实例中的 collectSqlQueries 参数设置为 true 以收集该实例中的调用。

启用全局 CollectSqlQueries 属性

以下示例显示如何为 .NET 和 .NET Core 启用 CollectSqlQueries 属性。

.NET

在 .NET 中您应用程序的全局配置内，要将 CollectSqlQueries 属性设置为 true，请修改您的 App.config 或 Web.config 文件的 appsettings，如图所示。

Example **App.config** 或 **Web.config** - 全局启用 SQL 查询的收集

```
<configuration>
<appSettings>
    <add key="CollectSqlQueries" value="true">
</appSettings>
```

```
</configuration>
```

.NET Core

在 .NET Core 中您应用程序的全局配置内，要将 `CollectSqlQueries` 属性设置为 `true`，请在 X-Ray 键下修改您的 `appsettings.json` 文件，如图所示。

Example **`appsettings.json`** - 全局启用 SQL 查询的收集

```
{
  "XRay": {
    "CollectSqlQueries":"true"
  }
}
```

启用该 collectSqlQueries 参数

您可以在 `TraceableSqlCommand` 实例中将 `collectSqlQueries` 参数设置为 `true`，以收集使用该实例进行的 SQL Server 查询的 SQL 查询文本。将参数设置为 `false` 禁用 `TraceableSqlCommand` 实例的 `CollectSqlQuery` 功能。

> **Note**
>
> `TraceableSqlCommand` 实例中 `collectSqlQueries` 的值将覆盖 `CollectSqlQueries` 属性的全局配置中设置的值。

Example 示例 **`Controller.cs`** - 启用实例的 SQL 查询收集

```
using Amazon;
using Amazon.Util;
using Amazon.XRay.Recorder.Core;
using Amazon.XRay.Recorder.Handlers.SqlServer;

private void QuerySql(int id)
{
  var connectionString = ConfigurationManager.AppSettings["RDS_CONNECTION_STRING"];
  using (var sqlConnection = new SqlConnection(connectionString))
  using (var command = new TraceableSqlCommand("SELECT " + id, sqlConnection,
 collectSqlQueries: true))
```

```
  {
    command.ExecuteNonQuery();
  }
}
```

创建附加子分段

子分段可为跟踪的分段扩展为了给请求提供服务而已完成的工作的详细信息。每次使用已检测的客户端进行调用时，X-Ray SDK 在子分段中记录生成的信息。您可以创建其他子分段来分组其他子分段，来度量某个代码段的性能如何，或是来记录注释和元数据。

要管理子分段，请使用 `BeginSubsegment` 和 `EndSubsegment` 方法。在 `try` 代码块的子分段中执行任何任务，使用 `AddException` 跟踪异常。在 `finally` 代码块中调用 `EndSubsegment` 确保结束子分段。

Example Controller.cs – 自定义子分段

```
AWSXRayRecorder.Instance.BeginSubsegment("custom method");
try
{
  DoWork();
}
catch (Exception e)
{
  AWSXRayRecorder.Instance.AddException(e);
}
finally
{
  AWSXRayRecorder.Instance.EndSubsegment();
}
```

当您在分段或者其他子分段中创建子分段时，适用于 .NET 的 X-Ray 开发工具包将为其生成 ID 并记录开始时间和结束时间。

Example 包含元数据的子分段

```
"subsegments": [{
  "id": "6f1605cd8a07cb70",
  "start_time": 1.480305974194E9,
  "end_time": 1.4803059742E9,
  "name": "Custom subsegment for UserModel.saveUser function",
```

```
  "metadata": {
    "debug": {
      "test": "Metadata string from UserModel.saveUser"
    }
  },
```

使用 X-Ray SDK for .NET，将注释和元数据添加到分段

可以利用注释和元数据记录与请求、环境或应用程序相关的其他信息。可以将注释和元数据添加到 X-Ray 开发工具包创建的分段或您创建的自定义子分段。

注释是带字符串、数字或布尔值的键值对。系统会对注释编制索引，以便与筛选表达式一起使用。使用注释记录要用于对控制台中的跟踪进行分组的数据或在调用 `GetTraceSummaries` API 时使用的数据。

元数据是可以具有任何类型值的键-值对，包括对象和列表，但没有编制索引，无法与筛选条件表达式一起使用。使用元数据记录要存储在跟踪中但不需要用于搜索跟踪的其他数据。

Sections

- 使用 X-Ray SDK for .NET 记录注释
- 使用 X-Ray SDK for .NET 记录元数据

使用 X-Ray SDK for .NET 记录注释

使用注释记录有关要为其编制索引以进行搜索的分段和子分段的信息。

X-Ray 中的所有注释都需要以下内容：

注释要求

- 键 - X-Ray 注释的键最多可以包含 500 个字母数字字符。除了点或句点（.）之外，不能使用空格或符号
- 值 - X-Ray 注释的值最多可以包含 1,000 个 Unicode 字符。
- 注释的数量 - 每个跟踪最多可使用 50 条注释。

在 AWS Lambda 函数之外录制注释

1. 获取 `AWSXRayRecorder` 的实例。

```
using Amazon.XRay.Recorder.Core;
...
AWSXRayRecorder recorder = AWSXRayRecorder.Instance;
```

2. 使用字符串键和布尔型、Int32、Int64、双精度或字符串值调用 addAnnotation。

```
recorder.AddAnnotation("mykey", "my value");
```

以下示例说明如何使用包含点和布尔值、数字值或字符串值的字符串键调用 putAnnotation。

```
document.putAnnotation("testkey.test", "my value");
```

在 AWS Lambda 函数内部录制注释

Lambda 函数中的分段和子分段均由 Lambda 运行时环境管理。如果要在 Lambda 函数中为分段或子分段添加注释，则必须执行以下操作：

1. 在 Lambda 函数中创建分段或子分段。
2. 将注释添加到分段或子分段。
3. 结束分段或子分段。

以下代码示例显示如何在 Lambda 函数中将注释添加到子分段：

```
#Create the subsegment
AWSXRayRecorder.Instance.BeginSubsegment("custom method");
#Add an annotation
AWSXRayRecorder.Instance.AddAnnotation("My", "Annotation");
try
{
  YourProcess(); #Your function
}
catch (Exception e)
{
  AWSXRayRecorder.Instance.AddException(e);
}
finally #End the subsegment
{
  AWSXRayRecorder.Instance.EndSubsegment();
```

```
}
```

X-Ray SDK 将注释以键-值对的形式记录在分段文档的 annotations 对象中。使用相同键调用两次 addAnnotation 操作将覆盖同一分段或子分段上之前记录的值。

要查找具有带特定值的注释的跟踪，请在annotation[*key*]筛选表达式中使用 关键字。

使用 X-Ray SDK for .NET 记录元数据

使用元数据可记录有关您无需编制索引即可在搜索中使用的分段或子分段的信息。元数据值可以是字符串、数字、布尔值或可序列化为 JSON 对象或数组的任何其他对象。

记录元数据

1. 获取 AWSXRayRecorder 的实例，如以下代码示例中所示：

   ```
   using Amazon.XRay.Recorder.Core;
   ...
   AWSXRayRecorder recorder = AWSXRayRecorder.Instance;
   ```

2. 使用字符串命名空间、字符串键和对象值调用 AddMetadata，如以下代码示例所示：

   ```
   recorder.AddMetadata("my namespace", "my key", "my value");
   ```

   也可以仅使用键和值对来调用 AddMetadata 操作，如以下代码示例中所示：

   ```
   recorder.AddMetadata("my key", "my value");
   ```

如果您没有指定命名空间的值，X-Ray SDK 将使用 default。使用相同键调用两次 AddMetadata 操作将覆盖同一分段或子分段上之前记录的值。

使用 Ruby

有两种方法可用于检测 Ruby 应用程序，以将跟踪数据发送到 X-Ray：

- AWS Distro for OpenTelemetry Ruby — 提供一组开源库的 AWS 发行版，用于通过 Distro for Collect AWS o r Collector 向多个 AWS 监控解决方案（包括亚马逊和亚马逊 OpenSearch 服务）发送相关的指标和跟踪。 CloudWatch AWS X-Ray OpenTelemetry
- AWS X-Ray 适用于 Ruby 的 SDK — 一组库，用于通过 X-Ray 守护程序生成跟踪并将其发送到 X-Ray。

有关更多信息，请参阅 在 AWS Distro for 和 X-Ray OpenTelemetry 之间进行选择 SDKs。

AWS Ruby 发行 OpenTelemetry 版

使用适用于 OpenTelemetry (ADOT) Ruby 的 AWS Distro，您可以对应用程序进行一次检测，然后将相关的指标和跟踪发送到多个 AWS 监控解决方案 CloudWatch，包括亚马逊 AWS X-Ray和亚马逊服务。 OpenSearch 将 X-Ray 与 ADOT 配合使用需要两个组件：启用 OpenTelemetry SDK 以与 X-Ray 配合使用，以及启用 Collecto OpenTelemetryr 的AWS 发行版以与 X-Ray 配合使用。

要开始使用，请参阅 OpenTelemetry Ruby AWS 发行版文档。

有关将 Distro 用于 with AWS X-Ray 和 other 的 OpenTelemetry 更多信息 AWS 服务，请参阅 AWS Distro for OpenTelemetry 或 AWS Distro for AWS D ocuments。 OpenTelemetry

有关语言支持和用法的更多信息，请参阅上的 O AWS bservability。 GitHub

AWS X-Ray 适用于 Ruby 的 SDK

X-Ray SDK 是一个面向 Ruby Web 应用程序的库，该库提供用于生成跟踪数据并将其发送到 X-Ray 进程守护程序的类和方法。跟踪数据包括有关应用程序处理的传入 HTTP 请求的信息，以及应用程序使用 AWS SDK、HTTP 客户端或活动记录客户端对下游服务进行的调用的信息。您还可以手动创建分段并在注释和元数据中添加调试信息。

您可以通过将 SDK 添加到 gemfile 并运行 `bundle install` 来下载 SDK。

Example Gemfile

```
gem 'aws-sdk'
```

如果您使用了 Rails，请首先添加 X-Ray SDK 中间件来跟踪传入请求。请求筛选器将创建一个分段。当分段打开时，您可以使用开发工具包客户端的方法将信息添加到分段，并创建子分段以跟踪下游调用。开发工具包还会自动记录在分段打开时应用程序引发的异常。对于非 Rails 应用程序，您可以手动创建分段。

接下来，使用 X-Ray SDK 通过配置记录器来修补关联的库，从而检测您 适用于 Ruby 的 AWS SDK 的、HTTP 和 SQL 客户端。每当您使用已检测的客户端调用下游 AWS 服务 或资源时，SDK 都会在子分段中记录有关该调用的信息。 AWS 服务 您在服务中访问的资源将作为下游节点显示在跟踪地图上，以帮助您识别各个连接上的错误和限制问题。

在开始使用 SDK 后，通过配置记录器来自定义其行为。您可以添加插件来记录有关运行应用程序的计算资源的数据，通过定义采样规则来自定义采样行为，提供记录器以在应用程序日志中查看来自 SDK 的更多或更少的信息。

记录有关请求以及应用程序在注释和元数据中所做的工作的其他信息。注释是简单的键值对，已为这些键值对编制索引以用于筛选条件表达式，以便您能够搜索包含特定数据的跟踪。元数据条目的限制性较低，并且可以记录整个对象和数组 - 可序列化为 JSON 的任何项目。

> **注释和元数据**
>
> 注释和元数据是您使用 X-Ray 开发工具包添加到分段的任意文本。系统会对注释编制索引，以便与筛选表达式一起使用。元数据未编制索引，但可以使用 X-Ray 控制台或 API 在原始分段中查看。您授予 X-Ray 读取权限的任何人都可以查看这些数据。

当代码中具有大量检测的客户端时，一个请求分段可包含大量子分段，检测的客户端发起的每个调用均对应一个子分段。您可以通过将客户端调用包含在自定义子分段中来整理子分段并为其分组。您可以为整个函数或任何代码部分创建自定义子分段，并记录子分段的元数据和注释，而不是编写父分段的所有内容。

有关 SDK 的类和方法的参考文档，请参阅 AWS X-Ray SDK for Ruby API 参考。

要求

X-Ray SDK 需要 Ruby 2.3 或更高版本，并且与以下库兼容：

- 适用于 Ruby 的 AWS SDK 3.0 或更高版本
- Rails 版本 5.1 或更高版本

配置适用于 Ruby 的 X-Ray 开发工具包

适用于 Ruby 的 X-Ray 开发工具包具有提供全局记录器的、名为 `XRay.recorder` 的类。您可以配置全局记录器以自定义为传入 HTTP 调用创建分段的中间件。

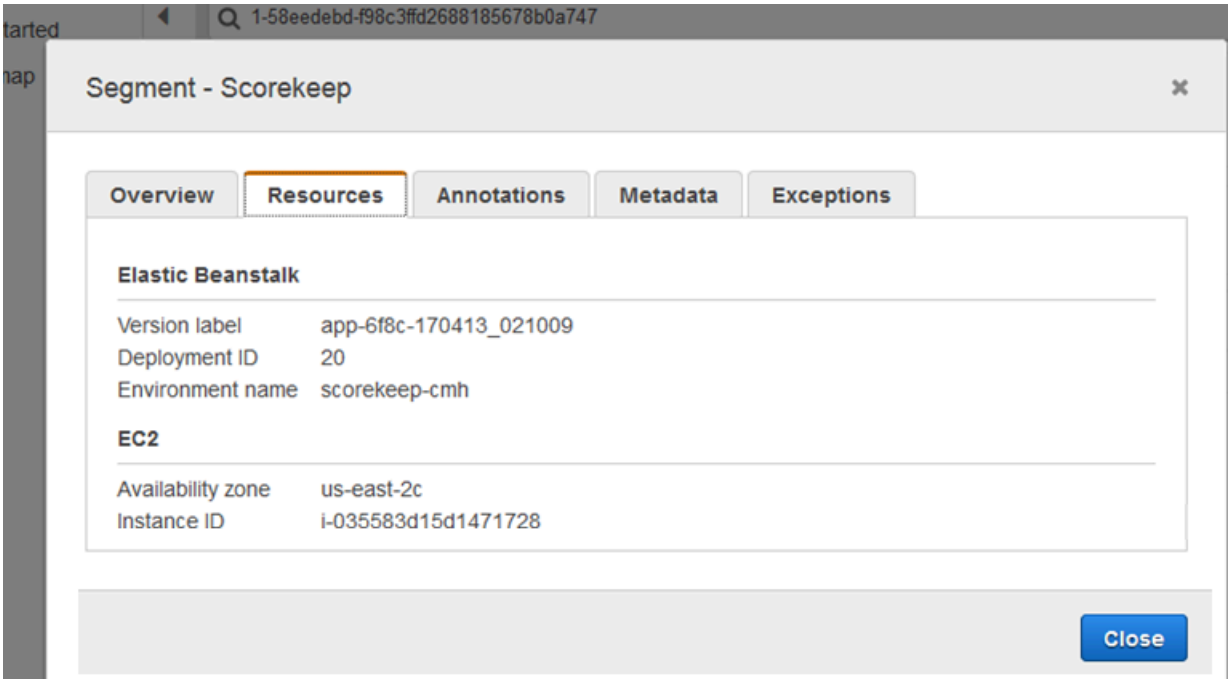
Sections

服务插件

`plugins` 用于记录有关托管应用程序的服务的信息。

插件

- Amazon EC2 — `ec2` 添加实例 ID 和可用区域。
- Elastic Beanstalk - `elastic_beanstalk` 添加环境名称、版本标签和部署 ID。
- Amazon ECS — `ecs` 添加容器 ID。



要使用插件，请在您传递给记录器的配置对象中指定插件。

Example main.rb - 插件配置

```
my_plugins = %I[ec2 elastic_beanstalk]

config = {
  plugins: my_plugins,
  name: 'my app',
}

XRay.recorder.configure(config)
```

您还可以使用环境变量来配置记录器，它优先于在代码中设置的值。

开发工具包还使用插件设置为设置分段上的 `origin` 字段。这表示运行您的应用程序的 AWS 资源类型。当您使用多个插件时，SDK 使用以下解析顺序来确定来源： ElasticBeanstalk > EKS > ECS > EC2。

采样规则

该 SDK 使用您在 X-Ray 控制台中定义的采样规则来确定要记录的请求。默认规则跟踪每秒的第一个请求，以及所有将跟踪发送到 X-Ray 的服务的任何其他请求的百分之五。在 X-Ray 控制台中创建其他规则以自定义为每个应用程序记录的数据量。

该 SDK 按照定义的顺序应用自定义规则。如果请求与多个自定义规则匹配，则 SDK 仅应用第一条规则。

> **Note**
>
> 如果 SDK 无法访问 X-Ray 来获取采样规则，它将恢复为默认的本地规则，即每秒第一个请求以及每个主机所有其他请求的百分之五。如果主机无权调用采样，或者无法连接到 X-Ray 守护程序 APIs，后者充当 SDK 发出的 API 调用的 TCP 代理，则可能会发生这种情况。

您还可以将 SDK 配置为从 JSON 文档加载采样规则。在 X-Ray 采样不可用的情况下，SDK 可以使用本地规则作为备份，也可以只使用本地规则。

Example sampling-rules.json

```
{
  "version": 2,
  "rules": [
    {
      "description": "Player moves.",
      "host": "*",
      "http_method": "*",
      "url_path": "/api/move/*",
      "fixed_target": 0,
      "rate": 0.05
    }
  ],
  "default": {
    "fixed_target": 1,
    "rate": 0.1
  }
}
```

此示例定义了一个自定义规则和一个默认规则。自定义规则采用百分之五的采样率，对于 `/api/move/` 之下的路径要跟踪的请求数量不设下限。默认规则中每秒的第一个请求以及其他请求的百分之十。

在本地定义规则的缺点是，固定目标由记录器的每个实例独立应用而不是由 X-Ray 服务管理。随着您部署更多主机，固定速率会成倍增加，这使得控制记录的数据量变得更加困难。

要配置备份规则，请在传递给记录器的配置对象中为文档定义一个哈希。

Example main.rb - 备份规则配置

```
require 'aws-xray-sdk'
my_sampling_rules =  {
  version: 1,
  default: {
    fixed_target: 1,
    rate: 0.1
  }
}
config = {
  sampling_rules: my_sampling_rules,
  name: 'my app',
}
XRay.recorder.configure(config)
```

要单独存储采样规则，请在一个单独的文件中定义哈希，并要求该文件将哈希拉入您的应用程序中。

Example config/sampling-rules.rb

```
my_sampling_rules =  {
  version: 1,
  default: {
    fixed_target: 1,
    rate: 0.1
  }
}
```

Example main.rb - 文件中的采样规则

```
require 'aws-xray-sdk'
require 'config/sampling-rules.rb'

config = {
  sampling_rules: my_sampling_rules,
  name: 'my app',
}
XRay.recorder.configure(config)
```

要仅使用本地规则，需要采样规则和配置 `LocalSampler`。

Example main.rb - 本地规则采样

```
require 'aws-xray-sdk'
require 'aws-xray-sdk/sampling/local/sampler'

config = {
  sampler: LocalSampler.new,
  name: 'my app',
}
XRay.recorder.configure(config)
```

您还可以配置全局记录器，以禁止对所有传入请求进行采样和检测。

Example main.rb - 禁用采样

```
require 'aws-xray-sdk'
config = {
  sampling: false,
  name: 'my app',
}
XRay.recorder.configure(config)
```

日志记录

默认情况下，记录器将信息级别事件输出到 `$stdout`。您可以通过在传递给记录器的配置对象中定义记录器来自定义日志记录。

Example main.rb - 日志记录

```
require 'aws-xray-sdk'
config = {
  logger: my_logger,
  name: 'my app',
}
XRay.recorder.configure(config)
```

当您手动生成子分段时，使用调试日志来识别诸如未结束子分段之类的问题。

代码中的记录器配置

其他设置包含在 `XRay.recorder` 的 `configure` 方法中。

- `context_missing` - 设置为 `LOG_ERROR` 可避免在您的已检测代码尝试在分段未打开的情况下记录数据时引发异常。
- `daemon_address` - 设置 X-Ray 进程守护程序侦听器的主机和端口。
- `name` - 设置开发工具包用于进行分段的服务名称。
- `naming_pattern` - 设置域名模式以使用动态命名。
- `plugins` - 使用插件记录有关应用程序的 AWS 资源的信息。
- `sampling` 设置为 `false` 可禁用采样。
- `sampling_rules` - 设置包含您的采样规则的哈希。

Example main.py - 禁用缺少上下文异常

```
require 'aws-xray-sdk'
config = {
  context_missing: 'LOG_ERROR'
}

XRay.recorder.configure(config)
```

使用 Rails 时的记录器配置

如果您使用的是 Rails 框架，则可在 `app_root/initializers` 下的 Ruby 文件中配置全局记录器的选项。X-Ray 开发工具包支持对 Rails 使用其他配置键。

- `active_record` - 设置为 `true` 可记录 Active Record 数据库事务的子分段。

在名为 `Rails.application.config.xray` 的配置对象中配置可用设置。

Example config/initializers/aws_xray.rb

```
Rails.application.config.xray = {
  name: 'my app',
  patch: %I[net_http aws_sdk],
  active_record: true
}
```

环境变量

您可以使用环境变量来配置 Ruby 的 X-Ray 开发工具包。开发工具包支持以下变量：

- AWS_XRAY_TRACING_NAME - 设置 SDK 用于进行分段的服务名称。覆盖您根据 servlet 筛选器的分段命名策略设置的服务名称。
- AWS_XRAY_DAEMON_ADDRESS - 设置 X-Ray 进程守护程序侦听器的主机和端口。默认情况下，开发工具包会将跟踪数据发送到 127.0.0.1:2000。如果您已将进程守护程序配置为侦听不同端口或者进程守护程序在另一台主机上运行，则使用此变量。
- AWS_XRAY_CONTEXT_MISSING - 设置为 RUNTIME_ERROR 会在您的已检测代码尝试在分段未打开的情况下记录数据时引发异常。

  有效值

  - RUNTIME_ERROR— 引发运行时异常。
  - LOG_ERROR— 记录错误并继续（默认）。
  - IGNORE_ERROR— 忽略错误并继续。

  对于在未打开任何请求时运行的启动代码或者会生成新线程的代码，如果您尝试在其中使用检测过的客户端，则可能发生与缺失分段或子分段相关的错误。

环境变量覆盖在代码中设置的值。

使用 X-Ray SDK for Ruby 中间件跟踪传入请求

您可以使用 X-Ray SDK 来跟踪您的应用程序在亚马逊或亚马逊 EC2 ECS 中的 EC2 实例上提供的传入 HTTP 请求。 AWS Elastic Beanstalk

如果您使用的是 Rails，请使用 Rails 中间件检测传入 HTTP 请求。在您将中间件添加到应用程序并配置分段名称时，X-Ray SDK for Ruby 会为每个采样请求创建一个分段。由其他检测创建的任何分段成为请求级别分段的子分段，请求级别的分段提供有关 HTTP 请求和响应的信息。此信息包括请求的计时、方法和处置。

每个分段都有一个名称，用于在服务映射中标识您的应用程序。可以静态命名分段，也可以将 SDK 配置为根据传入请求中的主机标头对其进行动态命名。动态命名允许根据请求中的域名对跟踪进行分组，并且在名称不匹配预期模式时（例如，如果主机标头是伪造的）应用默认名称。

> **转发的请求**
>
> 如果负载均衡器或其他中间将请求转发到您的应用程序，X-Ray 会提取请求 X-Forwarded-For 标头中的客户端 IP 而非 IP 数据包中的源 IP。由于转发的请求记录的客户端 IP 可以伪造，因此不应信任。

在转发请求时，SDK 在分段中设置附加字段来指示此行为。如果分段包含设置为 `x_forwarded_for` 的字段 `true`，则从 HTTP 请求的 `X-Forwarded-For` 标头获取客户端 IP。

中间件使用包含以下信息的 `http` 块为每个传入请求创建一个分段：

- HTTP 方法 - GET、POST、PUT、DELETE 等。
- 客户端地址 - 发送请求的客户端的 IP 地址。
- 响应代码 - 已完成请求的 HTTP 响应代码。
- 时间 - 开始时间（收到请求时）和结束时间（发送响应时）。
- 用户代理 - 请求中的 `user-agent`。
- 内容长度 - 响应中的 `content-length`。

使用 Rails 中间件

要使用中间件，请将 gemfile 更新为包含所需的 railtie。

Example Gemfile - rails

```
gem 'aws-xray-sdk', require: ['aws-xray-sdk/facets/rails/railtie']
```

要使用中间件，您还必须使用在跟踪地图中表示应用程序的名称来配置记录器。

Example config/initializers/aws_xray.rb

```
Rails.application.config.xray = {
  name: 'my app'
}
```

手动检测代码

如果您未使用 Rails，请手动创建分段。您可以为每个传入的请求创建区段，也可以围绕已修补的 HTTP 或 AWS SDK 客户端创建区段，为录制器添加子分段提供上下文。

```
# Start a segment
segment = XRay.recorder.begin_segment 'my_service'
# Start a subsegment
subsegment = XRay.recorder.begin_subsegment 'outbound_call', namespace: 'remote'
```

```
# Add metadata or annotation here if necessary
my_annotations = {
  k1: 'v1',
  k2: 1024
}
segment.annotations.update my_annotations

# Add metadata to default namespace
subsegment.metadata[:k1] = 'v1'

# Set user for the segment (subsegment is not supported)
segment.user = 'my_name'

# End segment/subsegment
XRay.recorder.end_subsegment
XRay.recorder.end_segment
```

配置分段命名策略

AWS X-Ray 使用服务名称来标识您的应用程序，并将其与您的应用程序使用的其他应用程序、数据库 APIs、外部数据库和 AWS 资源区分开来。当 X-Ray SDK 为传入请求生成分段时，会将应用程序的服务名称记录在分段的名称字段中。

X-Ray SDK 可以用在 HTTP 请求标头中的 hostname 来命名分段。不过，此标头可以伪造，会导致服务地图中出现意料之外的节点。为防止 SDK 由于包含伪造的主机标头的请求而错误地命名分段，必须为传入请求指定一个默认名称。

如果应用程序为多个域的请求提供服务，则可以将 SDK 配置为使用动态命名策略以在分段名称中反映出这一点。动态命名策略允许 SDK 将主机名用于符合预期模式的请求，并将默认名称应用于不符合预期模式的请求。

例如，可能有一款应用程序为发送到三个子域的请求提供服务，分别为 `www.example.com`、`api.example.com` 和 `static.example.com`。可以使用格式 `*.example.com` 的动态命名策略以识别包含不同名称的子域的分段，服务地图上因此会显示三个服务节点。如果应用程序收到包含与该格式不匹配的 hostname 的请求，您将会在服务地图上看到第四个节点，以及您指定的回退名称。

要对所有请求分段使用相同名称，请在配置记录器时指定应用程序的名称，如前几节所示。

动态命名策略定义一个主机名应匹配的模式和一个在 HTTP 请求中的主机名与该模式不匹配时要使用的默认名称。要动态命名分段，请在 config 哈希中指定命名模式。

Example main.rb - 动态命名

```
config = {
  naming_pattern: '*mydomain*',
  name: 'my app',
}

XRay.recorder.configure(config)
```

您可以在模式中使用“*”来匹配任何字符串，或使用“?”来匹配任意单个字符。

> **Note**
>
> 您可以使用 AWS_XRAY_TRACING_NAME 环境变量覆盖您在代码中定义的默认服务名称。

修补库以检测下游调用

要检测下游调用，请使用适用于 Ruby 的 X-Ray 开发工具包修补您的应用程序使用的库。适用于 Ruby 的 X-Ray 开发工具包可以修补以下库。

支持的库

- net/http - 检测 HTTP 客户端。
- aws-sdk— 仪器 适用于 Ruby 的 AWS SDK 客户。

如果您使用已修补的库，适用于 Ruby 的 X-Ray 开发工具包会为调用创建子分段，并记录请求和响应中的信息。必须通过开发工具包中间件或对 XRay.recorder.begin_segment 的调用提供分段，以供开发工具包创建子分段。

要修补库，请在您传递给 X-Ray 记录器的配置对象中指定这些库。

Example main.rb - 修补库

```
require 'aws-xray-sdk'

config = {
  name: 'my app',
```

```
  patch: %I[net_http aws_sdk]
}

XRay.recorder.configure(config)
```

使用适用于 Ruby 的 X-Ray SD AWS K 追踪 SDK 调用

当您的应用调用 AWS 服务 以存储数据、写入队列或发送通知时，适用于 Ruby 的 X-Ruby 的 X-Ruby SDK 会按子分段跟踪下游的调用。在这些服务（例如，Amazon S3 存储桶或 Amazon SQS 队列）中追踪的资源 AWS 服务 和访问的资源在 X-Ray 控制台的跟踪地图上显示为下游节点。

当你修补aws-sdk库时，适用于 Ruby 的 X-Ruby AWS SDK 会自动检测所有 SDK 客户端。您无法检测单个客户端。

对于所有服务，都可以在 X-Ray 控制台中看到调用的 API 的名称。X-Ray 开发工具包会为一部分服务将信息添加到分段，从而在服务地图中提供更高的粒度。

例如，当使用经过检测的 DynamoDB 客户端发出调用时，对于针对表的调用，开发工具包会将表名称添加到分段中。在控制台中，每个表在服务地图中显示为一个独立的节点，以及没有表作为目标的调用的一般 DynamoDB 节点。

Example 对 DynamoDB 进行调用以保存项目的子分段

```
{
  "id": "24756640c0d0978a",
  "start_time": 1.480305974194E9,
  "end_time": 1.4803059742E9,
  "name": "DynamoDB",
  "namespace": "aws",
  "http": {
    "response": {
      "content_length": 60,
      "status": 200
    }
  },
  "aws": {
    "table_name": "scorekeep-user",
    "operation": "UpdateItem",
    "request_id": "UBQNSO5AEM8T4FDA4RQDEB94OVTDRVV4K4HIRGVJF66Q9ASUAAJG",
  }
}
```

在您访问指定的资源时，对以下服务的调用会在服务地图中创建额外的节点。没有定向到特定资源的调用，为服务创建了通用节点。

- Amazon DynamoDB - 表名称
- Amazon Simple Storage Service - 存储桶和键名称
- Amazon Simple Queue Service - 队列名称

使用 X-Ray 开发工具包生成自定义子分段

子分段可为跟踪的分段扩展为了给请求提供服务而已完成的工作的详细信息。每次使用已检测的客户端进行调用时，X-Ray SDK 在子分段中记录生成的信息。您可以创建其他子分段来分组其他子分段，来度量某个代码段的性能如何，或是来记录注释和元数据。

要管理子分段，请使用 `begin_subsegment` 和 `end_subsegment` 方法。

```
subsegment = XRay.recorder.begin_subsegment name: 'annotations', namespace: 'remote'
my_annotations = { id: 12345 }
subsegment.annotations.update my_annotations
XRay.recorder.end_subsegment
```

要为函数创建子分段，可将其包装在对 `XRay.recorder.capture` 的调用中。

```
XRay.recorder.capture('name_for_subsegment') do |subsegment|
  resp = myfunc() # myfunc is your function
  subsegment.annotations.update k1: 'v1'
  resp
end
```

当您在分段或者其他子分段中创建子分段时，X-Ray 开发工具包将为其生成 ID 并记录开始时间和结束时间。

Example 包含元数据的子分段

```
"subsegments": [{
  "id": "6f1605cd8a07cb70",
  "start_time": 1.480305974194E9,
  "end_time": 1.4803059742E9,
  "name": "Custom subsegment for UserModel.saveUser function",
  "metadata": {
    "debug": {
```

```
      "test": "Metadata string from UserModel.saveUser"
    }
  },
```

使用 X-Ray SDK for Ruby，将注释和元数据添加到分段

可以利用注释和元数据记录与请求、环境或应用程序相关的其他信息。可以将注释和元数据添加到 X-Ray 开发工具包创建的分段或您创建的自定义子分段。

注释是带字符串、数字或布尔值的键值对。系统会对注释编制索引，以便与筛选表达式一起使用。使用注释记录要用于对控制台中的跟踪进行分组的数据或在调用 `GetTraceSummaries` API 时使用的数据。

元数据是可以具有任何类型值的键-值对，包括对象和列表，但没有编制索引，无法与筛选条件表达式一起使用。使用元数据记录要存储在跟踪中但不需要用于搜索跟踪的其他数据。

除了注释和元数据之外，您还可以在分段上记录用户 ID 字符串。用户 IDs 被记录在区段的单独字段中，并编制索引以供搜索使用。

Sections

- 使用 X-Ray SDK for Ruby 记录注释
- 使用 X-Ray SDK for Ruby 记录元数据
- 使用适用于 Ruby IDs 的 X-Ray SDK 录制用户

使用 X-Ray SDK for Ruby 记录注释

使用注释记录有关要为其编制索引以进行搜索的分段和子分段的信息。

注释要求

- 键 - X-Ray 注释的键最多可以包含 500 个字母数字字符。除了点或句点（.）之外，不能使用空格或符号
- 值 - X-Ray 注释的值最多可以包含 1,000 个 Unicode 字符。
- 注释的数量 - 每个跟踪最多可使用 50 条注释。

记录注释

1. 从 `xray_recorder` 获取对当前分段或子分段的引用。

```
require 'aws-xray-sdk'
...
document = XRay.recorder.current_segment
```

或

```
require 'aws-xray-sdk'
...
document = XRay.recorder.current_subsegment
```

2. 调用带哈希值的 update。

```
my_annotations = { id: 12345 }
document.annotations.update my_annotations
```

以下示例演示如何使用包含点的注释调用 update。

```
my_annotations = { testkey.test: 12345 }
document.annotations.update my_annotations
```

开发工具包将注释以键-值对的形式记录在分段文档的 annotations 对象中。使用相同键调用两次 add_annotations 将覆盖同一分段或子分段上之前记录的值。

要查找具有带特定值的注释的跟踪，请在annotation[*key*]筛选表达式中使用 关键字。

使用 X-Ray SDK for Ruby 记录元数据

使用元数据记录有关您无需为其编制索引以进行搜索的分段或子分段的信息。元数据值可以是字符串、数字、布尔值或可序列化为 JSON 对象或数组的任何对象。

记录元数据

1. 从 xray_recorder 获取对当前分段或子分段的引用。

```
require 'aws-xray-sdk'
...
document = XRay.recorder.current_segment
```

或

```
require 'aws-xray-sdk'
...
document = XRay.recorder.current_subsegment
```

2. 使用字符串键、布尔值、数字、字符串或对象值以及字符串命名空间调用 metadata。

```
my_metadata = {
  my_namespace: {
    key: 'value'
  }
}
subsegment.metadata my_metadata
```

使用相同键调用两次 metadata 将覆盖同一分段或子分段上之前记录的值。

使用适用于 Ruby IDs 的 X-Ray SDK 录制用户

记录请求细分中的用户，以识别发送请求的用户。 IDs

要记录用户 IDs

1. 从 xray_recorder 获取对当前分段的引用。

```
require 'aws-xray-sdk'
...
document = XRay.recorder.current_segment
```

2. 将分段上的用户字段设置为发送请求的用户的字符串 ID。

```
segment.user = 'U12345'
```

您可以在控制器中设置用户以便在应用程序开始处理请求后立即记录用户 ID。

要查找用户 ID 的跟踪，请在user筛选表达式中使用 关键字。

从 X-Ray 仪器迁移到 OpenTelemetry 仪器

X-Ray 正在过渡到 OpenTelemetry (OTel) 作为其应用跟踪和可观察性的主要仪器标准。这种战略转变 AWS 符合行业最佳实践，为客户提供了更全面、更灵活和面向未来的解决方案，以满足他们的可观察性需求。 OpenTelemetry它在业界得到广泛采用，可以跨不同的系统追踪请求，包括 AWS 那些可能无法直接与 X-Ray 集成的外部系统。

本章为平稳过渡提供了建议，并强调了迁移到 OpenTelemetry基于解决方案的重要性，以确保持续支持和访问应用程序仪器和可观察性方面的最新功能。

建议将其 OpenTelemetry 用作检测应用程序的可观察性解决方案。

主题

理解 OpenTelemetry

OpenTelemetry 是一个行业标准的可观测性框架，它提供了用于收集遥测数据的标准化协议和工具。它提供了一种统一的方法来检测、生成、收集和导出遥测数据，例如指标、日志和跟踪。

从 X-Ray 迁移 SDKs 到时 OpenTelemetry，您将获得以下好处：

- 增强的框架和库工具支持
- Support 支持其他编程语言
- 自动仪表功能
- 灵活的采样配置选项

- 统一收集指标、日志和跟踪

与 X-Ray 守护程序相比，收集 OpenTelemetry 器为数据收集格式和导出目标提供了更多的选项。

OpenTelemetry 支持 AWS

AWS 为处理以下问题提供了多种解决方案 OpenTelemetry：

- AWS 发行版适用于 OpenTelemetry

  将 OpenTelemetry 轨迹作为分段导出到 X-Ray。

  有关更多信息，请参阅AWS 发行版。 OpenTelemetry
- CloudWatch 应用程序信号

  导出自定义 OpenTelemetry 跟踪和指标以监控应用程序运行状况。

  有关更多信息，请参阅使用应用程序信号。
- CloudWatch OTel 端点

  使用带有本机 OpenTelemetry 仪器的 HTTP OTel 端点将 OpenTelemetry 跟踪导出到 X-Ray。

  有关更多信息，请参阅使用 OTel 终端节点。

OpenTelemetry 与一起使用 AWS CloudWatch

AWS CloudWatch 支持 OpenTelemetry 通过客户端应用程序工具和本机 AWS CloudWatch 服务（例如应用程序信号、跟踪、地图、指标和日志）进行跟踪。有关更多信息，请参阅 OpenTelemetry。

了解迁移 OpenTelemetry 概念

下表将 X-Ray 概念映射到它们的 OpenTelemetry 等效概念。了解这些映射有助于您将现有的 X-Ray 仪器转换为 OpenTelemetry：

X 射线概念	OpenTelemetry 概念
X-Ray 记录器	示踪剂提供者和示踪剂
服务插件	资源探测器

X 射线概念	OpenTelemetry 概念
客户细分	（服务器）跨度
子细分市场	（非服务器）跨度
X 射线采样规则	OpenTelemetry 采样（可定制）
X-Ray 发射器	Span 导出器（可定制）
注释/元数据	Attributes
图书馆仪器	图书馆仪器
X-Ray 追踪背景	跨度上下文
X-Ray 轨迹上下文传播	W3C 跟踪上下文传播
X 射线痕迹采样	OpenTelemetry 轨迹采样
不适用	跨度处理
不适用	行李
X-Ray 守护程序	OpenTelemetry 收藏家

> **Note**
>
> 有关 OpenTelemetry 概念的更多信息，请参阅OpenTelemetry 文档。

比较功能

下表显示了这两种服务都支持哪些功能。使用此信息来确定迁移期间需要解决的任何差距：

功能	X 射线仪器	OpenTelemetry 仪器
图书馆仪器	支持	支持
X 射线采样	支持	在 OTel java/.net/Go 中支持

功能	X 射线仪器	OpenTelemetry 仪器
		在 ADOT Java/ 中支持。 NET/ Python/Node.js
X-Ray 轨迹上下文传播	支持	支持
资源检测	支持	支持
区段注释	支持	支持
区段元数据	支持	支持
零代码自动检测	在 Java 中支持	在 OTel Java/ 中支持。 NET/ Python/Node.js 在 ADOT Java/ 中支持。 NET/ Python/Node.js
手动创建跟踪	支持	支持

设置和配置跟踪

要在中创建跟踪 OpenTelemetry，你需要一个示踪剂。您可以通过在应用程序中初始化 Trac er 提供者来获得示踪器。这与使用 X-Recorder 配置 X-Recorder 以及在 X-Ray 轨迹中创建分段和子分段的方式类似。

> **Note**
>
> 与 X-Ray Record OpenTelemetry er 相比，Tracer Provider 提供了更多的配置选项。

了解轨迹数据结构

在了解了基本概念和功能映射之后，您可以了解特定的实现细节，例如跟踪数据结构和采样。

OpenTelemetry 使用跨度而不是分段和子分段来构造跟踪数据。每个跨度包括以下组件：

- 名称
- 唯一标识

- 开始和结束时间戳
- 跨度种类
- 跨越上下文
- 属性（键值元数据）
- 事件（带时间戳的日志）
- 指向其他跨度的链接
- 状态信息
- 父跨度引用

迁移到时 OpenTelemetry，您的跨度会自动转换为 X-Ray 分段或子分段。这样可以确保您现有的主 CloudWatch 机体验保持不变。

使用跨度属性

X-Ray SDK 提供了两种向区段和子分段添加数据的方法：

Annotations

为筛选和搜索编制索引的键值对

元数据

包含未编入索引以供搜索的复杂数据的键值对

默认情况下， OpenTelemetry 跨度属性会转换为 X-Ray 原始数据中的元数据。要改为将特定属性转换为注释，请将其键添加到`aws.xray.annotations`属性列表中。

- 有关 OpenTelemetry 概念的更多信息，请参阅OpenTelemetry 跟踪
- 有关 OpenTelemetry 数据如何映射到 X-Ray 数据的详细信息，请参阅 OpenTelemetry X-Ray 数据模型转换

检测环境中的资源

OpenTelemetry 使用资源检测器收集有关生成遥测数据的资源的元数据。此元数据存储为资源属性。例如，生成遥测数据的实体可以是 Amazon ECS 集群或 Amazon EC2 实例，可以从这些实体记录的资源属性可以包括 Amazon ECS 集群 ARN 或 EC2 亚马逊实例 ID。

- 有关支持的资源类型的信息，请参阅OpenTelemetry 资源语义约定

- 有关 X-Ray 服务插件的信息，请参阅配置 X-Ray SDK

管理抽样策略

跟踪采样通过从具有代表性的请求子集而不是所有请求中收集数据来帮助您管理成本。OpenTelemetry 和 X-Ray 都支持采样，但实现方式有所不同。

> **Note**
>
> 采样少于 100% 的轨迹可以降低可观察性成本，同时保持对应用程序性能的有意义的见解。

OpenTelemetry 提供了多种内置采样策略，并允许您创建自定义采样策略。您还可以使用某些 SDK 语言配置 X-Remote Sampler，以便使用 X-Ray 采样规则。 OpenTelemetry

来自的其他抽样策略 OpenTelemetry 有：

- 基于父母的抽样 — 在应用其他抽样策略之前，请尊重父跨度的抽样决定
- 基于跟踪 ID 比率的采样 — >随机抽取指定百分比的跨度
- 尾部采样 — 将采样规则应用于 OpenTelemetry 收集器中的完整轨迹
- 自定义采样器 — 使用采样接口实现自己的采样逻辑

有关 X-Ray 采样规则的信息，请参阅 X-Ra y 控制台中的采样规则

有关 OpenTelemetry 尾部采样的信息，请参见尾部采样处理器

管理追踪上下文

X-Ray SDKs 管理区段上下文，以正确处理追踪中区段和子区段之间的父子关系。 OpenTelemetry 使用类似的机制来确保跨度具有正确的父跨度。它在整个请求上下文中存储和传播跟踪数据。例如，当您的应用程序处理请求并创建服务器跨度来表示该请求时， OpenTelemetry 会将服务器跨度存储在 OpenTelemetry 上下文中，以便在创建子跨度时，该子跨度可以在上下文中引用该跨度作为其父跨度。

传播跟踪上下文

X-Ray 和 HTTP 标头都 OpenTelemetry 使用 HTTP 标头跨服务传播跟踪上下文。这使您可以链接不同服务生成的跟踪数据并维护采样决策。

X-Ray SDK 使用 X-Ray 跟踪标头自动传播跟踪上下文。当一个服务调用另一个服务时，跟踪标头包含维护跟踪之间父子关系所需的上下文。

OpenTelemetry 支持用于上下文传播的多种跟踪标头格式，包括：

- W3C 跟踪上下文（默认）
- X-Ray 追踪标头
- 其他自定义格式

> **Note**
>
> 您可以配置 OpenTelemetry 为使用一种或多种标题格式。例如，使用 X-Ray Propagator 向支持 X-Ray 跟踪的 AWS 服务发送跟踪上下文。

配置并使用 X-Ray Propagator 启用跨 AWS 服务的跟踪。这允许您将跟踪上下文传播到 API Gateway 端点和其他支持 X-Ray 的服务。

- 有关 X-Ray 跟踪标头的信息，请参阅 X-Ray 开发人员指南中的跟踪标头
- 有关 OpenTelemetry 上下文传播的信息，请参阅 OpenTelemetry 文档中的上下文和上下文传播

使用图书馆工具

X-Ray 和 X-Ray 都 OpenTelemetry 提供了库工具，只需最少的代码更改即可向应用程序添加跟踪。

X-Ray 提供图书馆仪器功能。这允许您添加预先构建的 X-Ray 仪器，而只需对应用程序代码进行最少的更改。这些工具支持特定的库，例如 AWS SDK 和 HTTP 客户端，以及 Spring Boot 或 Express.js 等网络框架。

OpenTelemetry的工具库通过库挂钩或自动代码修改为您的库生成详细的跨度，只需最少的代码更改。

要确定 OpenTelemetry's Library Instrumentations是否支持您的库，请在 OpenTelemetry 注册表的注册表中进行搜索。OpenTelemetry

导出轨迹

X-Ray 并 OpenTelemetry 使用不同的方法导出跟踪数据。

X-Ray 轨迹导出

X-Ray SDKs 使用发射器发送轨迹数据：

- 向 X-Ray 守护程序发送分段和子分段
- 使用 UDP 进行非阻塞 I/O
- 在 SDK 中默认配置

OpenTelemetry 追踪导出

OpenTelemetry 使用可配置的 Span 导出器发送跟踪数据：

- 使用 http/protobuf 或 grpc 协议
- 将跨度导出到由 OpenTelemetry 收集器或 CloudWatch 代理监控的端点
- 允许自定义导出器配置

处理和转发跟踪

X-Ray 和都 OpenTelemetry 提供用于接收、处理和转发跟踪数据的组件。

X 射线追踪处理

X-Ray Daemon 处理跟踪处理：

- 监听来自 X-Ray 的 UDP 流量 SDKs
- 批量区段和子细分
- 将批量上传到 X-Ray 服务

OpenTelemetry 跟踪处理

OpenTelemetry 收集器处理追踪处理：

- 接收来自仪器化服务的跟踪
- 处理并可选择修改跟踪数据
- 将处理过的跟踪发送到各种后端，包括 X-Ray

> **Note**
>
> AWS CloudWatch 代理还可以接收 OpenTelemetry 轨迹并将其发送到 X-Ray。有关更多信息，请参阅使用收集指标和跟踪 OpenTelemetry。

跨度处理（OpenTelemetry特定概念）

OpenTelemetry 使用跨度处理器在创建跨度时对其进行修改：

- 允许在创建或完成时读取和修改跨度
- 启用跨度处理的自定义逻辑

行李（OpenTelemetry特定概念）

OpenTelemetry的行李功能允许传播键值数据：

- 允许在跟踪上下文旁边传递任意数据
- 对于跨服务边界传播特定于应用程序的信息很有用

有关 OpenTelemetry 收集器的信息，请参见OpenTelemetry 收集器

有关 X-Ray 概念的信息，请参阅《X-Ray 开发者指南》中的 X-Ray 概念

迁移概述

本节概述了迁移所需的代码更改。以下列表是特定语言的指导和 X-Ray Daemon 迁移步骤。

> **Important**
>
> 要从 X-Ray 仪器完全迁移到 OpenTelemetry 仪器，您需要：
>
> 1. 用 OpenTelemetry 解决方案取代 X-Ray SDK 的使用
> 2. 将 X-Ray 守护程序替换为 CloudWatch 代理或 OpenTelemetry 收集器（使用 X-Ray 导出器）

- 迁移到 OpenTelemetry Java

- 迁移到 OpenTelemetry Go
- 迁移到 OpenTelemetry Node.js
- 迁移到 OpenTelemetry .NET
- 迁移到 OpenTelemetry Python
- 迁移到 OpenTelemetry Ruby

针对新应用程序和现有应用程序的建议

对于新的和现有的应用程序，建议使用以下解决方案在应用程序中启用跟踪：

Instrumentation

- OpenTelemetry SDKs
- AWS 仪器仪表发行 OpenTelemetry 版

数据收集

- OpenTelemetry 收藏家
- CloudWatch 代理人

迁移到 OpenTelemetry基于基础的解决方案后，您的 CloudWatch 体验将保持不变。您仍然可以在 CloudWatch 控制台的 Traces 和 Trace Map 页面中以相同格式查看您的踪迹，或者通过 X- Ray 检索您的跟踪数据 APIs。

跟踪设置更改

您需要用设置替换 X-Ray OpenTelemetry 设置。

X-Ray 和 OpenTelemetry 设置的比较

功能	X-ray SDK	OpenTelemetry
默认配置	• X-Ray 集中采样 • X-Ray 轨迹上下文传播 • 跟踪导出到 X-Ray 守护程序	• 将跟踪导出到 OpenTelemetry 收集器或 CloudWatch 代理 (http/gRPC) • W3C 跟踪上下文传播
手动配置	• 本地抽样规则 • 资源检测插件	• X-Ray 采样（可能不适用于所有语言）

功能	X-ray SDK	OpenTelemetry
		• 资源检测 • X-Ray 轨迹上下文传播

图书馆工具变更

更新您的代码，使用 OpenTelemetry 库工具代替 AWS SDK、HTTP 客户端、Web 框架和其他库的 X-Ray 库工具。这会生成 OpenTelemetry 轨迹而不是 X-Ray 轨迹。

> **Note**
>
> 代码更改因语言和库而异。有关详细说明，请参阅特定语言的迁移指南。

Lambda 环境工具变更

要 OpenTelemetry 在 Lambda 函数中使用，请选择以下设置选项之一：

1. 使用自动插入 Lambda 层：
   - （推荐）CloudWatch 应用程序信号 Lambda 层

     > **Note**
     >
     > 要仅使用跟踪，请设置 Lambda 环境变量。`OTEL_AWS_APPLICATION_SIGNALS_ENABLED=false`

   - AWS 适用于 ADOT 的托管 Lambda 层
2. 手动设置您 OpenTelemetry 的 Lambda 函数：
   - 使用 X-Ray UDP 跨度导出器配置简单跨度处理器
   - 设置 X-Ray Lambda 传播器

手动创建跟踪数据

将 X-Ray 分段和子分段替换为 OpenTelemetry Span：

- 使用示 OpenTelemetry 踪剂创建跨度

- 向 Span 添加属性（等同于 X-Ray 元数据和注释）

> **Important**
>
> 发送到 X-Ray 时：
>
> - 服务器跨度转换为 X-Ray 片段
> - 其他跨度转换为 X-Ray 子分段
> - 默认情况下，属性会转换为元数据

要将属性转换为注释，请将其密钥添加到`aws.xray.annotations`属性列表中。有关更多信息，请参阅启用自定义 X-Ray 注释。

从 X-Ray Daemon 迁移到 AWS CloudWatch 代理或收集器 OpenTelemetry

您可以使用 CloudWatch 代理或 OpenTelemetry 采集器从装有仪器的应用程序接收跟踪并将其发送到 X-Ray。

> **Note**
>
> CloudWatch 代理版本 1.300025.0 及更高版本可以收集跟踪。 OpenTelemetry 使用 CloudWatch 代理代替 X-Ray Daemon 可以减少需要管理的代理数量。有关更多信息，请参阅使用 CloudWatch 代理收集指标、日志和跟踪。

Sections

- 在 Amazon EC2 或本地服务器上迁移
- 在 Amazon ECS 上迁移
- 在 Elastic Beanstalk 上迁移

在 Amazon EC2 或本地服务器上迁移

> **Important**
>
> 在使用 CloudWatch 代理或 OpenTelemetry 收集器之前，请停止 X-Ray Daemon 进程，以防止端口冲突。

现有的 X-Ray 守护程序设置

安装守护程序

您现有的 X-Ray Daemon 用法是使用以下方法之一安装的：

手动安装

从 X-Ray 守护程序 Amazon S3 存储桶下载并运行可执行文件。

自动安装

启动实例时，使用此脚本安装守护程序：

```
#!/bin/bash
curl https://s3.us-east-2.amazonaws.com/aws-xray-assets.us-east-2/xray-daemon/aws-xray-daemon-3.x.rpm \
    -o /home/ec2-user/xray.rpm
yum install -y /home/ec2-user/xray.rpm
```

配置 进程守护程序

您现有的 X-Ray Daemon 使用是使用以下任一方法配置的：

- 命令行参数
- 配置文件 (`xray-daemon.yaml`)

Example 使用配置文件

```
./xray -c ~/xray-daemon.yaml
```

运行进程守护程序

您现有的 X-Ray Daemon 使用是通过以下命令开始的：

```
~/xray-daemon$ ./xray -o -n us-east-1
```

正在移除守护程序

要从您的亚马逊 EC2 实例中删除 X-Ray Daemon，请执行以下操作：

1. 停止守护程序服务：

   ```
   systemctl stop xray
   ```

2. 删除配置文件：

   ```
   rm ~/path/to/xray-daemon.yaml
   ```

3. 如果已配置，请删除日志文件：

   > **Note**
   >
   > 日志文件位置取决于您的配置：
   >
   > - 命令行配置：`/var/log/xray-daemon.log`
   > - 配置文件：检查`LogPath`设置

设置代 CloudWatch 理

安装座席

有关安装说明，请参阅在本地服务器上安装 CloudWatch 代理。

配置代理

1. 创建配置文件以启用跟踪收集。有关更多信息，请参阅创建 CloudWatch 代理配置文件。
2. 设置 IAM 权限：
   - 为代理附加 IAM 角色或指定证书。有关更多信息，请参阅设置 IAM 角色。
   - 确保一个或多个角色凭证包含`xray:PutTraceSegments`权限。

启动 代理

有关启动代理的说明，请参阅使用命令行启动 CloudWatch 代理。

设置 OpenTelemetry 收集器

安装收集器

下载并安装适用于您的操作系统的 OpenTelemetry 收集器。有关说明，请参阅安装收集器。

配置收集器

在收集器中配置以下组件：

- awsproxy 扩展

  X-Ray 采样所必需的
- OTel 接收器

  从您的应用程序中收集痕迹
- X 射线导出器

  向 X-Ray 发送轨迹

Example 采集器配置示例 — otel-collector-config .yaml

```
extensions:
  awsproxy:
    endpoint: 127.0.0.1:2000
  health_check:

receivers:
  otlp:
    protocols:
      grpc:
        endpoint: 127.0.0.1:4317
      http:
        endpoint: 127.0.0.1:4318

processors:
  batch:
```

```
exporters:
  awsxray:
    region: 'us-east-1'

service:
  pipelines:
    traces:
      receivers: [otlp]
      exporters: [awsxray]
  extensions: [awsproxy, health_check]
```

> **Important**
>
> 使用`xray:PutTraceSegments`权限配置 AWS 凭证。有关更多信息，请参阅指定凭据。

启动收集器

使用您的配置文件运行收集器：

```
otelcol --config=otel-collector-config.yaml
```

在 Amazon ECS 上迁移

> **Important**
>
> 您的任务角色必须拥有您使用的任何收集器的`xray:PutTraceSegments`权限。
> 在同一台主机上运行 CloudWatch 代理或 OpenTelemetry 收集器容器之前，请停止任何现有的 X-Ray Daemon 容器，以防止端口冲突。

使用代 CloudWatch 理

1. 从亚马逊 ECR 公共画廊获取 Docker 图片。
2. 创建名为`cw-agent-otel.json`：的配置文件

```
{
  "traces": {
    "traces_collected": {
```

```
      "xray": {
        "tcp_proxy": {
          "bind_address": "0.0.0.0:2000"
        }
      },
      "otlp": {
        "grpc_endpoint": "0.0.0.0:4317",
        "http_endpoint": "0.0.0.0:4318"
      }
    }
  }
}
```

3. 将配置存储在 Systems Manager 参数存储中：

   1. 打开 https://console.aws.amazon.com/systems-manager/
   2. 选择 “创建参数”
   3. 输入以下值：
      - 名称: `/ecs/cwagent/otel-config`
      - 等级：标准
      - 类型：字符串
      - 数据类型：文本
      - 值：[将 cw-agent-otel .json 配置粘贴到此处]
4. 使用桥接网络模式创建任务定义：

   在您的任务定义中，配置取决于您使用的联网模式。桥式联网是默认模式，可在您的默认 VPC 中使用。在桥接网络中，设置`OTEL_EXPORTER_OTLP_TRACES_ENDPOINT`环境变量以告诉 OpenTelemetry SDK CloudWatch 代理的终端节点和端口。您还应该创建一个从应用程序容器到 Collector 容器的链接，以便将跟踪从应用程序中的 OpenTelemetry SDK 发送到 Collector 容器。

   Example CloudWatch 代理任务定义

```
{
    "containerDefinitions": [
        {
            "name": "cwagent",
            "image": "public.ecr.aws/cloudwatch-agent/cloudwatch-agent:latest",
            "portMappings": [
                {
                    "containerPort": 4318,
```

```
                    "hostPort": 4318,
                    "protocol": "tcp"
                },
                {
                    "containerPort": 4317,
                    "hostPort": 4317,
                    "protocol": "tcp"
                },
                {
                    "containerPort": 2000,
                    "hostPort": 2000,
                    "protocol": "tcp"
                }
            ],
            "secrets": [
                {
                    "name": "CW_CONFIG_CONTENT",
                    "valueFrom": "/ecs/cwagent/otel-config"
                }
            ]
        },
        {
            "name": "application",
            "image": "APPLICATION_IMAGE",
            "links": ["cwagent"],
            "environment": [
                {
                    "name": "OTEL_EXPORTER_OTLP_TRACES_ENDPOINT",
                    "value": "http://cwagent:4318/v1/traces"
                }
            ]
        }
    ]
}
```

有关更多信息，请参阅在 Amazon ECS 上部署 CloudWatch 代理以收集 Amazon EC2 实例级指标。

使用 OpenTelemetry 收集器

1. `otel/opentelemetry-collector-contrib`从 Docker Hub 获取 Docker 镜像。
2. `otel-collector-config.yaml`使用与 Amazon 配置收集器部分相同的内容创建名为的 EC2 配置文件，但要更新终端节点以`0.0.0.0`代替`127.0.0.1`。

3. 要在 Amazon ECS 中使用此配置，您可以将配置存储在 Systems Manager 参数存储中。首先，前往 Systems Manager 参数存储控制台，然后选择创建新参数。使用以下信息创建新参数：
   - 名称：/ecs/otel/config（收集器的任务定义中将引用此名称）
   - 等级：标准
   - 类型：字符串
   - 数据类型：文本
   - 值：[将 otel-collector-config .yaml 配置粘贴到此处]
4. 以桥接网络模式为例，创建部署 OpenTelemetry 收集器的任务定义。

   在任务定义中，配置取决于您使用的联网模式。桥式联网是默认模式，可在您的默认 VPC 中使用。在桥接网络中，设置OTEL_EXPORTER_OTLP_TRACES_ENDPOINT环境变量以告诉 S OpenTelemetry DK OpenTelemetry 收集器的端点和端口。您还应该创建一个从应用程序容器到 Collector 容器的链接，以便将跟踪从应用程序中的 OpenTelemetry SDK 发送到 Collector 容器。

   Example OpenTelemetry 收集器任务定义

```
{
    "containerDefinitions": [
        {
            "name": "otel-collector",
            "image": "otel/opentelemetry-collector-contrib",
            "portMappings": [
                {
                    "containerPort": 2000,
                    "hostPort": 2000
                },
                {
                    "containerPort": 4317,
                    "hostPort": 4317
                },
                {
                    "containerPort": 4318,
                    "hostPort": 4318
                }
            ],
            "command": [
                "--config",
                "env:SSM_CONFIG"
            ],
            "secrets": [
```

```
                {
                    "name": "SSM_CONFIG",
                    "valueFrom": "/ecs/otel/config"
                }
            ]
        },
        {
            "name": "application",
            "image": "APPLICATION_IMAGE",
            "links": ["otel-collector"],
            "environment": [
                {
                    "name": "OTEL_EXPORTER_OTLP_TRACES_ENDPOINT",
                    "value": "http://otel-collector:4318/v1/traces"
                }
            ]
        }
    ]
}
```

在 Elastic Beanstalk 上迁移

> **Important**
>
> 在使用 CloudWatch 代理之前停止 X-Ray Daemon 进程，以防止端口冲突。

您现有的 X-Ray Daemon 集成是通过使用 Elastic Beanstalk 控制台开启的，或者通过使用配置文件在应用程序源代码中配置 X-Ray Daemon。

使用代 CloudWatch 理

在 Amazon Linux 2 平台上，使用`.ebextensions`配置文件配置 CloudWatch 代理：

1. 在项目根目录`.ebextensions`中创建一个名为的目录
2. 在`.ebextensions`目录`cloudwatch.config`中创建一个名为的文件，内容如下：

```
files:
  "/opt/aws/amazon-cloudwatch-agent/etc/config.json":
    mode: "0644"
```

```
    owner: root
    group: root
    content: |
      {
        "traces": {
          "traces_collected": {
            "otlp": {
              "grpc_endpoint": "12.0.0.1:4317",
              "http_endpoint": "12.0.0.1:4318"
            }
          }
        }
      }
container_commands:
  start_agent:
    command: /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a
 append-config -c file:/opt/aws/amazon-cloudwatch-agent/etc/config.json -s
```

3. 部署时将该.ebextensions目录包含在应用程序源包中

有关 Elastic Beanstalk 配置文件的更多信息，请参阅使用配置文件进行高级环境自定义。

迁移到 OpenTelemetry Java

本节提供有关从 X-Ray SDK 迁移到适用于 Java 应用程序的 OpenTelemetry SDK 的指导。

Sections

- 零代码自动检测解决方案
- 使用 SDK 的手动仪器解决方案
- 跟踪传入的请求（Spring 框架工具）
- AWS 软件开发工具包 v2 插件
- 检测传出 HTTP 调用
- 对其他库的仪器支持
- 手动创建跟踪数据
- Lambda 仪器

零代码自动检测解决方案

With X-Ray Java agent

要启用 X-Ray Java 代理，需要修改应用程序的 JVM 参数。

```
-javaagent:/path-to-disco/disco-java-agent.jar=pluginPath=/path-to-disco/disco-
plugins
```

With OpenTelemetry-based Java agent

使用 OpenTelemetry基于 Java 的代理。

- 使用 AWS Distro for OpenTelemetry (ADOT) Auto-Instrumention Java 代理使用 ADOT Java 代理进行自动检测。有关更多信息，请参阅使用 Java 代理自动检测跟踪和指标。如果您只想跟踪，请禁用`OTEL_METRICS_EXPORTER=none` 环境变量. 以从 Java 代理导出指标。

  （可选）您还可以在使用 ADOT Java 自动检测应用程序时启用 CloudWatch 应用程序信号，以监控当前应用程序的运行状况并跟踪长期应用程序性能。 AWS Application Signals 提供以应用程序为中心的统一视图，了解您的应用程序、服务和依赖关系，并帮助监控和分类应用程序的运行状况。有关更多信息，请参阅 Application Signals。
- 使用 OpenTelemetry Java 代理进行自动检测。有关更多信息，请参阅使用 Java 代理进行零代码检测。

使用 SDK 的手动仪器解决方案

Tracing setup with X-Ray SDK

要使用适用于 Java 的 X-Ray SDK 来检测您的代码，首先需要为该`AWSXRay`类配置服务插件和本地采样规则，然后使用提供的记录器。

```
static { AWS XRayRecorderBuilder builder = AWS
 XRayRecorderBuilder.standard().withPlugin(new EC2Plugin()).withPlugin(new
 ECSPlugin()); AWS XRay.setGlobalRecorder(builder.build());
}
```

Tracing setup with OpenTelemetry SDK

以下依赖关系是必需的。

```
<dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>io.opentelemetry</groupId>
                <artifactId>opentelemetry-bom</artifactId>
                <version>1.49.0</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
            <dependency>
                <groupId>io.opentelemetry.instrumentation</groupId>
                <artifactId>opentelemetry-instrumentation-bom</artifactId>
                <version>2.15.0</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>io.opentelemetry</groupId>
            <artifactId>opentelemetry-sdk</artifactId>
        </dependency>
        <dependency>
            <groupId>io.opentelemetry</groupId>
            <artifactId>opentelemetry-api</artifactId>
        </dependency>
        <dependency>
            <groupId>io.opentelemetry.semconv</groupId>
            <artifactId>opentelemetry-semconv</artifactId>
        </dependency>
        <dependency>
            <groupId>io.opentelemetry</groupId>
            <artifactId>opentelemetry-exporter-otlp</artifactId>
        </dependency>
        <dependency>
            <groupId>io.opentelemetry.contrib</groupId>
            <artifactId>opentelemetry-aws-xray</artifactId>
            <version>1.46.0</version>
        </dependency>
        <dependency>
            <groupId>io.opentelemetry.contrib</groupId>
            <artifactId>opentelemetry-aws-xray-propagator</artifactId>
```

```
            <version>1.46.0-alpha</version>
        </dependency>
        <dependency>
            <groupId>io.opentelemetry.contrib</groupId>
            <artifactId>opentelemetry-aws-resources</artifactId>
            <version>1.46.0-alpha</version>
        </dependency>
    </dependencies>
```

通过实例化来配置 OpenTelemetry SDK，`TracerProvider`然后全局注册一个对象。`OpenTelemetrySdk`配置以下组件：

- OTLP Span 导出器（例如 OtlpGrpcSpanExporter）-将跟踪导出到 CloudWatch 代理或 OpenTelemetry 收集器时必需的
- AWS X-Ray 传播器 — 将跟踪上下文传播到与 X-Ray 集成的 AWS 服务所必需的
- AWS X-Ray-Remote 采样器 — 如果您需要使用 X-Ray 采样规则对请求进行采样，则需要使用
- 资源检测器（例如， EcsResource 或 Ec2Resource）— 检测运行应用程序的主机的元数据

```
import io.opentelemetry.api.common.Attributes;
import io.opentelemetry.context.propagation.ContextPropagators;
import io.opentelemetry.contrib.aws.resource.Ec2Resource;
import io.opentelemetry.contrib.aws.resource.EcsResource;
import io.opentelemetry.contrib.awsxray.AwsXrayRemoteSampler;
import io.opentelemetry.contrib.awsxray.propagator.AwsXrayPropagator;
import io.opentelemetry.exporter.otlp.trace.OtlpGrpcSpanExporter;
import io.opentelemetry.sdk.OpenTelemetrySdk;
import io.opentelemetry.sdk.resources.Resource;
import io.opentelemetry.sdk.trace.SdkTracerProvider;
import io.opentelemetry.sdk.trace.export.BatchSpanProcessor;
import io.opentelemetry.sdk.trace.samplers.Sampler;
import static io.opentelemetry.semconv.ServiceAttributes.SERVICE_NAME;

// ...

    private static final Resource otelResource =
        Resource.create(Attributes.of(SERVICE_NAME, "YOUR_SERVICE_NAME"))
            .merge(EcsResource.get())
            .merge(Ec2Resource.get());
    private static final SdkTracerProvider sdkTracerProvider =
        SdkTracerProvider.builder()
            .addSpanProcessor(BatchSpanProcessor.create(
                OtlpGrpcSpanExporter.getDefault()
```

```
            ))
            .addResource(otelResource)
            .setSampler(Sampler.parentBased(
                AwsXrayRemoteSampler.newBuilder(otelResource).build()
            ))
            .build();
   // Globally registering a TracerProvider makes it available throughout the
application to create as many Tracers as needed.
   private static final OpenTelemetrySdk openTelemetry =
       OpenTelemetrySdk.builder()
           .setTracerProvider(sdkTracerProvider)

.setPropagators(ContextPropagators.create(AwsXrayPropagator.getInstance()))
           .buildAndRegisterGlobal();
```

跟踪传入的请求（Spring 框架工具）

With X-Ray SDK

有关如何使用带有 Spring 框架的 X-Ray SDK 来检测您的应用程序的信息，请参阅带有 Sprin g 的 AOP 和适用于 Java 的 X-Ray SDK。要在春季启用 AOP，请完成以下步骤。

1. 配置 Spring
2. 向应用程序添加跟踪过滤器
3. 为代码添加注释或实现接口
4. 激活应用程序中的 X-Ray

With OpenTelemetry SDK

OpenTelemetry 提供工具库来收集对 Spring Boot 应用程序的传入请求的跟踪。要以最少的配置启用 Spring Boot 插桩工具，请添加以下依赖项。

```
<dependency>
          <groupId>io.opentelemetry.instrumentation</groupId>
           <artifactId>opentelemetry-spring-boot-starter</artifactId>
       </dependency>
```

有关如何为您的 OpenTelemetry 设置启用和配置 Spring Boot 工具 OpenTelemetry的更多信息，请参阅入门。

Using OpenTelemetry-based Java agents

对 Spring Boot 应用程序进行检测的默认推荐方法是使用带有字节码检测的 OpenTelemetry Java 代理，与直接使用 SDK 相比，它还提供了更多的 out-of-the-box检测和配置。有关入门的信息，请参阅零代码自动检测解决方案。

AWS 软件开发工具包 v2 插件

With X-Ray SDK

当你在版本中添加`aws-xray-recorder-sdk-aws-sdk-v2-instrumentor`子模块时，适用于 Java 的 X-Ray AWS SDK 可以自动检测所有 SDK v2 客户端。

要使用 AWS SDK for Java 2.2 及更高版本检测单个客户端对 AWS 服务的下游客户端调用，您的编译配置中已排除该`aws-xray-recorder-sdk-aws-sdk-v2`模块，并包含该模块。`aws-xray-recorder-sdk-aws-sdk-v2-instrumentor` 通过配置单个客户端来对其进行检测。`TracingInterceptor`

```
import com.amazonaws.xray.interceptors.TracingInterceptor;
import software.amazon.awssdk.core.client.config.ClientOverrideConfiguration
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
//...

public class MyModel {
  private DynamoDbClient client = DynamoDbClient.builder()
    .region(Region.US_WEST_2)
    .overrideConfiguration(
      ClientOverrideConfiguration.builder()
        .addExecutionInterceptor(new TracingInterceptor())
        .build()
      )
    .build();
//...
```

With OpenTelemetry SDK

要自动检测所有 AWS SDK 客户端，请添加`opentelemetry-aws-sdk-2.2-autoconfigure`子模块。

```
<dependency>
```

```
            <groupId>io.opentelemetry.instrumentation</groupId>
            <artifactId>opentelemetry-aws-sdk-2.2-autoconfigure</artifactId>
            <version>2.15.0-alpha</version>
            <scope>runtime</scope>
        </dependency>
```

要检测单个 AWS SDK 客户端，请添加opentelemetry-aws-sdk-2.2子模块。

```
<dependency>
            <groupId>io.opentelemetry.instrumentation</groupId>
            <artifactId>opentelemetry-aws-sdk-2.2</artifactId>
            <version>2.15.0-alpha</version>
            <scope>compile</scope>
        </dependency>
```

然后，在创建 AWS SDK 客户端时注册拦截器。

```
import io.opentelemetry.instrumentation.awssdk.v2_2.AwsSdkTelemetry;

// ...

    AwsSdkTelemetry telemetry = AwsSdkTelemetry.create(openTelemetry);
    private final S3Client S3_CLIENT = S3Client.builder()
      .overrideConfiguration(ClientOverrideConfiguration.builder()
        .addExecutionInterceptor(telemetry.newExecutionInterceptor())
        .build())
      .build();
```

检测传出 HTTP 调用

With X-Ray SDK

要使用 X-Ray 检测传出的 HTTP 请求，需要适用于 Java 的 Apache HttpClient 版本的 X-Ray SDK。

```
import com.amazonaws.xray.proxies.apache.http.HttpClientBuilder;
...
  public String randomName() throws IOException {
    CloseableHttpClient httpclient = HttpClientBuilder.create().build();
```

With OpenTelemetry SDK

与 X-Ray Java SDK `ApacheHttpClientTelemetry` 类似，它 OpenTelemetry 提供了一个具有生成器方法的类，该方法允许创建 the 的实例，`HttpClientBuilder`以便为 A HttpClient pache 提供 OpenTelemetry基于跨度和上下文传播。

```
<dependency>
            <groupId>io.opentelemetry.instrumentation</groupId>
            <artifactId>opentelemetry-apache-httpclient-5.2</artifactId>
            <version>2.15.0-alpha</version>
            <scope>compile</scope>
        </dependency>
```

以下是来自的代码示例opentelemetry-java-instrumentation。 newHttpClient() 提供的 HTTP 客户端将为已执行的请求生成跟踪。

```
import io.opentelemetry.api.OpenTelemetry;
import
 io.opentelemetry.instrumentation.apachehttpclient.v5_2.ApacheHttpClientTelemetry;
import org.apache.hc.client5.http.classic.HttpClient;
import org.apache.hc.client5.http.impl.classic.HttpClientBuilder;

public class ApacheHttpClientConfiguration {

  private OpenTelemetry openTelemetry;

  public ApacheHttpClientConfiguration(OpenTelemetry openTelemetry) {
    this.openTelemetry = openTelemetry;
  }

  // creates a new http client builder for constructing http clients with open
 telemetry instrumentation
  public HttpClientBuilder createBuilder() {
    return
 ApacheHttpClientTelemetry.builder(openTelemetry).build().newHttpClientBuilder();
  }

  // creates a new http client with open telemetry instrumentation
  public HttpClient newHttpClient() {
    return ApacheHttpClientTelemetry.builder(openTelemetry).build().newHttpClient();
  }
}
```

对其他库的仪器支持

在 “支持的库、框架、应用程序服务器和” 下的相应工具 GitHub 库中查找支持 OpenTelemetry Java 的库工具的完整列表。 JVMs

或者，您可以搜索 OpenTelemetry 注册表以了解是否 OpenTelemetry 支持检测。要开始搜索，请参阅注册表。

手动创建跟踪数据

With X-Ray SDK

使用 X-Ray SDK，需要使用`beginSegment`和`beginSubsegment`方法来手动创建 X-Ray 分段和子分段。

```
 Segment segment = xrayRecorder.beginSegment("ManualSegment");
       segment.putAnnotation("annotationKey", "annotationValue");
       segment.putMetadata("metadataKey", "metadataValue");

       try {
           Subsegment subsegment =
xrayRecorder.beginSubsegment("ManualSubsegment");
           subsegment.putAnnotation("key", "value");

           // Do something here

       } catch (Exception e) {
           subsegment.addException(e);
       } finally {
           xrayRecorder.endSegment();
       }
```

With OpenTelemetry SDK

您可以使用自定义跨度来监控未被仪器库捕获的内部活动的性能。请注意，只有跨度种类服务器会转换为 X-Ray 分段，所有其他跨度都会转换为 X-Ray 子分段。

首先，你需要创建一个 Tracer 来生成跨度，你可以通过该`openTelemetry.getTracer`方法获得跨度。这将提供示例中全局注册`TracerProvider`的 Tracer 实使用 SDK 的手动仪器解决方案例。您可以根据需要创建任意数量的 Tracer 实例，但通常为整个应用程序使用一个 Tracer。

```
Tracer tracer = openTelemetry.getTracer("my-app");
```

你可以使用 Tracer 来创建跨度。

```
import io.opentelemetry.api.common.AttributeKey;
import io.opentelemetry.api.trace.Span;
import io.opentelemetry.api.trace.SpanKind;
import io.opentelemetry.api.trace.Tracer;
import io.opentelemetry.context.Scope;

...

// SERVER span will become an X-Ray segment
Span span = tracer.spanBuilder("get-token")
  .setKind(SpanKind.SERVER)
  .setAttribute("key", "value")
  .startSpan();
try (Scope ignored = span.makeCurrent()) {

  span.setAttribute("metadataKey", "metadataValue");
  span.setAttribute("annotationKey", "annotationValue");

  // The following ensures that "annotationKey: annotationValue" is an annotation in
 X-Ray raw data.
  span.setAttribute(AttributeKey.stringArrayKey("aws.xray.annotations"),
 List.of("annotationKey"));

  // Do something here
}

span.end();
```

跨度的默认类型为 “内部”。

```
// Default span of type INTERNAL will become an X-Ray subsegment
Span span = tracer.spanBuilder("process-header")
  .startSpan();
try (Scope ignored = span.makeCurrent()) {
  doProcessHeader();
}
```

使用 OpenTelemetry SDK 向跟踪添加注释和元数据

在上面的示例中，该`setAttribute`方法用于向每个跨度添加属性。默认情况下，所有 span 属性都将转换为 X-Ray 原始数据中的元数据。为了确保将属性转换为注释而不是元数据，上面的示例

将该属性的键添加到aws.xray.annotations属性列表中。有关更多信息，请参见启用自定义 X-Ray 注释和注释以及元数据。

使用 OpenTelemetry基于 Java 的代理

如果您使用 Java 代理自动检测应用程序，则需要在应用程序中执行手动插入。例如，在应用程序中检测任何自动仪器库未涵盖的部分的代码。

要使用代理执行手动检测，您需要使用该opentelemetry-api 构件。对象版本不能比代理版本更新。

```
import io.opentelemetry.api.GlobalOpenTelemetry;
import io.opentelemetry.api.trace.Span;

// ...

        Span parentSpan = Span.current();
        Tracer tracer = GlobalOpenTelemetry.getTracer("my-app");
        Span span = tracer.spanBuilder("my-span-name")
            .setParent(io.opentelemetry.context.Context.current().with(parentSpan))
            .startSpan();
        span.end();
```

Lambda 仪器

With X-Ray SDK

使用 X-Ray SDK，在您的 Lambda 启用主动跟踪后，无需进行任何其他配置即可使用 X-Ray SDK。Lambda 将创建一个表示 Lambda 处理程序调用的区段，您无需任何额外配置即可使用 X-Ray SDK 创建子分段或仪器库。

With OpenTelemetry-based solutions

自动检测 Lambda 层 — 您可以使用以下解决方案自动检测 AWS 带有凸起的 Lambda 层的 Lambda 层：

- CloudWatch 应用程序信号 Lambda 层（推荐）

> **Note**
>
> 此 Lambda 层默认启用 CloudWatch 应用程序信号，通过收集指标和跟踪来监控您的 Lambda 应用程序的性能和运行状况。为了仅进行跟踪，请设置 Lambda 环境变量。`OTEL_AWS_APPLICATION_SIGNALS_ENABLED=false`

  - 为您的 Lambda 应用程序启用性能和运行状况监控
  - 默认情况下会同时收集指标和跟踪
- AWS 适用于 ADOT Java 的托管 Lambda 层。有关更多信息，请参阅适用于 Java 的 L AWS amb OpenTelemetry da Support 发行版。

要将手动检测与自动检测层一起使用，请参见使用 SDK 的手动仪器解决方案。为了减少冷启动，可以考虑使用 OpenTelemetry 手动检测为您的 Lambda 函数生成 OpenTelemetry 跟踪。

OpenTelemetry 针对 AWS Lambda 的手动检测

考虑以下用于调用 Amazon S3 ListBuckets 的 Lambda 函数代码。

```
package example;

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

import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.ListBucketsRequest;
import software.amazon.awssdk.services.s3.model.ListBucketsResponse;
import software.amazon.awssdk.services.s3.model.S3Exception;

public class ListBucketsLambda implements RequestHandler<String, String> {

    private final S3Client S3_CLIENT = S3Client.builder()
        .build();

    @Override
    public String handleRequest(String input, Context context) {
        try {
            ListBucketsResponse response = makeListBucketsCall();
            context.getLogger().log("response: " + response.toString());
```

```
            return "Success";
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }

    private ListBucketsResponse makeListBucketsCall() {
        try {
            ListBucketsRequest listBucketsRequest = ListBucketsRequest.builder()
                .build();
            ListBucketsResponse response = S3_CLIENT.listBuckets(listBucketsRequest);
            return response;
        } catch (S3Exception e) {
            throw new RuntimeException("Failed to call S3 listBuckets" +
 e.awsErrorDetails().errorMessage(), e);
        }
    }
}
```

以下是依赖关系。

```
dependencies {
    implementation('com.amazonaws:aws-lambda-java-core:1.2.3')
    implementation('software.amazon.awssdk:s3:2.28.29')
    implementation('org.slf4j:slf4j-nop:2.0.16')
}
```

要手动检测您的 Lambda 处理程序和 Amazon S3 客户端，请执行以下操作。

1. 将实现RequestHandler（或 RequestStreamHandler）的函数类替换为扩展TracingRequestHandler（或 TracingRequestStreamHandler）的函数类。
2. 实例化一个对象 TracerProvider 并全局注册一个对象。 OpenTelemetrySdk 建议配置为：TracerProvider

    a. 带有 X-Ray UDP 跨度导出器的简单跨度处理器，用于向 Lambda 的 UDP X-Ray 端点发送跟踪

    b. ParentBased 始终开启的采样器（如果未配置，则为默认值）

    c. 将 service.name 设置为 Lambda 函数名称的资源

    d. X-Ray Lambda 传播器
3. 将handleRequest方法更改为doHandleRequest并将OpenTelemetrySdk对象传递给基类。

4. 通过在构建客户端时注册拦截器，使用 OpenTemetry AWS 软件开发工具包检测 Amazon S3 客户端。

你需要以下 OpenTelemetry相关的依赖关系。

```
dependencies {
    ...

    implementation("software.amazon.distro.opentelemetry:aws-distro-opentelemetry-xray-udp-span-exporter:0.1.0")

    implementation(platform('io.opentelemetry.instrumentation:opentelemetry-instrumentation-bom:2.14.0'))
    implementation(platform('io.opentelemetry:opentelemetry-bom:1.48.0'))

    implementation('io.opentelemetry:opentelemetry-sdk')
    implementation('io.opentelemetry:opentelemetry-api')
    implementation('io.opentelemetry.contrib:opentelemetry-aws-xray-propagator:1.45.0-alpha')
    implementation('io.opentelemetry.contrib:opentelemetry-aws-resources:1.45.0-alpha')
    implementation('io.opentelemetry.instrumentation:opentelemetry-aws-lambda-core-1.0:2.14.0-alpha')
    implementation('io.opentelemetry.instrumentation:opentelemetry-aws-sdk-2.2:2.14.0-alpha')
}
```

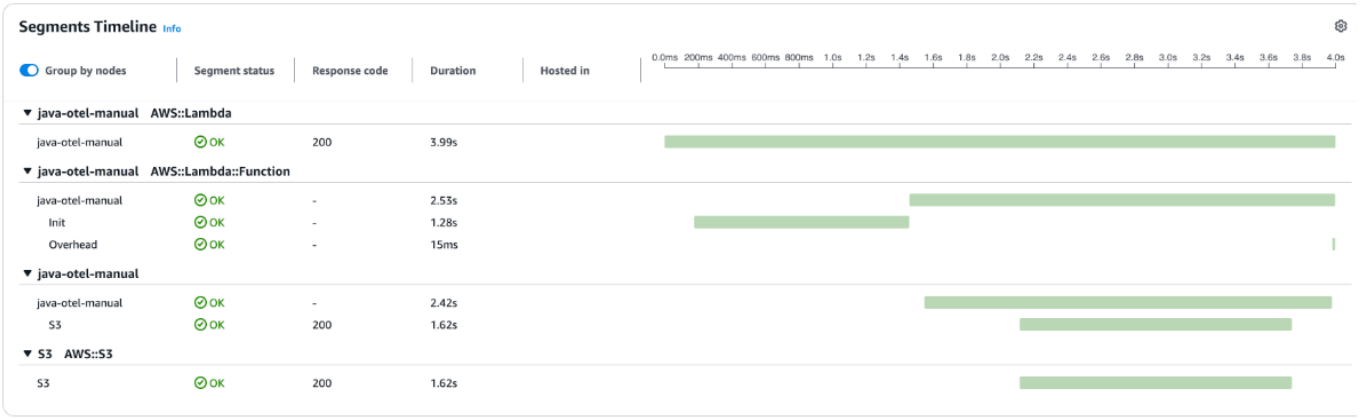
以下代码演示了进行必要更改后的 Lambda 函数。您可以创建其他自定义跨度来补充自动提供的跨度。

```
package example;

import java.time.Duration;

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

import io.opentelemetry.api.common.Attributes;
import io.opentelemetry.context.propagation.ContextPropagators;
import io.opentelemetry.contrib.aws.resource.LambdaResource;
import io.opentelemetry.contrib.awsxray.propagator.AwsXrayLambdaPropagator;
import io.opentelemetry.instrumentation.awslambdacore.v1_0.TracingRequestHandler;
import io.opentelemetry.instrumentation.awssdk.v2_2.AwsSdkTelemetry;
import io.opentelemetry.sdk.OpenTelemetrySdk;
```

```
import io.opentelemetry.sdk.resources.Resource;
import io.opentelemetry.sdk.trace.SdkTracerProvider;
import io.opentelemetry.sdk.trace.export.SimpleSpanProcessor;
import io.opentelemetry.sdk.trace.samplers.Sampler;
import static io.opentelemetry.semconv.ServiceAttributes.SERVICE_NAME;
import software.amazon.awssdk.core.client.config.ClientOverrideConfiguration;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.ListBucketsRequest;
import software.amazon.awssdk.services.s3.model.ListBucketsResponse;
import software.amazon.awssdk.services.s3.model.S3Exception;
import
 software.amazon.distro.opentelemetry.exporter.xray.udp.trace.AwsXrayUdpSpanExporterBuilder;

public class ListBucketsLambda extends TracingRequestHandler<String, String> {
    private static final Resource lambdaResource = LambdaResource.get();
    private static final SdkTracerProvider sdkTracerProvider =
        SdkTracerProvider.builder()
            .addSpanProcessor(SimpleSpanProcessor.create(
                new AwsXrayUdpSpanExporterBuilder().build()
            ))
            .addResource(
                lambdaResource
                .merge(Resource.create(Attributes.of(SERVICE_NAME,
 System.getenv("AWS_LAMBDA_FUNCTION_NAME"))))
            )
            .setSampler(Sampler.parentBased(Sampler.alwaysOn()))
            .build();
    private static final OpenTelemetrySdk openTelemetry =
        OpenTelemetrySdk.builder()
            .setTracerProvider(sdkTracerProvider)

 .setPropagators(ContextPropagators.create(AwsXrayLambdaPropagator.getInstance()))
            .buildAndRegisterGlobal();
    private static final AwsSdkTelemetry telemetry =
 AwsSdkTelemetry.create(openTelemetry);
    private final S3Client S3_CLIENT = S3Client.builder()
        .overrideConfiguration(ClientOverrideConfiguration.builder()
            .addExecutionInterceptor(telemetry.newExecutionInterceptor())
            .build())
        .build();

    public ListBucketsLambda() {
        super(openTelemetry, Duration.ofMillis(0));
    }
```

```
    @Override
    public String doHandleRequest(String input, Context context) {
        try {
            ListBucketsResponse response = makeListBucketsCall();
            context.getLogger().log("response: " + response.toString());
            return "Success";
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }

    private ListBucketsResponse makeListBucketsCall() {
        try {
            ListBucketsRequest listBucketsRequest = ListBucketsRequest.builder()
                .build();
            ListBucketsResponse response = S3_CLIENT.listBuckets(listBucketsRequest);
            return response;
        } catch (S3Exception e) {
            throw new RuntimeException("Failed to call S3 listBuckets" +
 e.awsErrorDetails().errorMessage(), e);
        }
    }
}
```

调用 Lambda 函数时，您将在控制台的 Trace Map 下看到以下跟踪。 CloudWatch



迁移到 OpenTelemetry Go

从 X-Ray 迁移时，使用以下代码示例使用 OpenTelemetry SDK 手动检测您的 Go 应用程序。

使用 SDK 进行手动检测

Tracing setup with X-Ray SDK

使用 X-Ray SDK for Go 时，需要先配置服务插件或本地采样规则，然后才能对代码进行检测。

```
func init() {
    if os.Getenv("ENVIRONMENT") == "production" {
        ec2.Init()
    }

    xray.Configure(xray.Config{
        DaemonAddr:       "127.0.0.1:2000",
        ServiceVersion:   "1.2.3",
    })
}
```

Set up tracing with OpenTelemetry SDK

通过实例化 a TracerProvider 并将其注册为全局跟踪器提供程序来配置 OpenTelemetry SDK。我们建议配置以下组件：

- OTLP 跟踪导出器 — 向 CloudWatch 代理或 OpenTelemetry 收集器导出跟踪时需要此项
- X-Ray Propagator — 需要将跟踪上下文传播到与 X-Ray 集成的 AWS 服务
- X-Ray-Remote Sampler — 使用 X-Ray 采样规则进行采样请求时需要使用
- 资源检测器-检测运行应用程序的主机的元数据

```
import (
    "go.opentelemetry.io/contrib/detectors/aws/ec2"
    "go.opentelemetry.io/contrib/propagators/aws/xray"
    "go.opentelemetry.io/contrib/samplers/aws/xray"
    "go.opentelemetry.io/otel"
    "go.opentelemetry.io/otel/exporters/otlp/otlptrace/otlptracegrpc"
    "go.opentelemetry.io/otel/sdk/trace"
)

func setupTracing() error {
    ctx := context.Background()

    exporterEndpoint := os.Getenv("OTEL_EXPORTER_OTLP_ENDPOINT")
    if exporterEndpoint == "" {
        exporterEndpoint = "localhost:4317"
    }

    traceExporter, err := otlptracegrpc.New(ctx,
        otlptracegrpc.WithInsecure(),
        otlptracegrpc.WithEndpoint(exporterEndpoint))
    if err != nil {
        return fmt.Errorf("failed to create OTLP trace exporter: %v", err)
    }

    remoteSampler, err := xray.NewRemoteSampler(ctx, "my-service-name", "ec2")
    if err != nil {
        return fmt.Errorf("failed to create X-Ray Remote Sampler: %v", err)
    }

    ec2Resource, err := ec2.NewResourceDetector().Detect(ctx)
```

```
    if err != nil {
        return fmt.Errorf("failed to detect EC2 resource: %v", err)
    }

    tp := trace.NewTracerProvider(
        trace.WithSampler(remoteSampler),
        trace.WithBatcher(traceExporter),
        trace.WithResource(ec2Resource),
    )

    otel.SetTracerProvider(tp)
    otel.SetTextMapPropagator(xray.Propagator{})

    return nil
}
```

跟踪传入的请求（HTTP 处理程序检测）

With X-Ray SDK

为了使用 X-Ray 对 HTTP 处理程序进行检测，使用了 X-Ray 处理程序方法来生成分段 NewFixedSegmentNamer。

```
func main() {
    http.Handle("/", xray.Handler(xray.NewFixedSegmentNamer("myApp"),
 http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
        w.Write([]byte("Hello!"))
    })))
    http.ListenAndServe(":8000", nil)
}
```

With OpenTelemetry SDK

要使用 HTTP 处理程序进行检测 OpenTelemetry，请使用 OpenTelemetry的 newHandler 方法来封装您的原始处理程序代码。

```
import (
```

```
    "go.opentelemetry.io/contrib/instrumentation/net/http/otelhttp"
)

helloHandler := func(w http.ResponseWriter, req *http.Request) {
    ctx := req.Context()
    span := trace.SpanFromContext(ctx)
    span.SetAttributes(attribute.Bool("isHelloHandlerSpan", true),
 attribute.String("attrKey", "attrValue"))

    _, _ = io.WriteString(w, "Hello World!\n")
}

otelHandler := otelhttp.NewHandler(http.HandlerFunc(helloHandler), "Hello")

http.Handle("/hello", otelHandler)
err = http.ListenAndServe(":8080", nil)
if err != nil {
    log.Fatal(err)
}
```

AWS 适用于 Go v2 插桩的 SDK

With X-Ray SDK

为了检测来自 AWS SDK 的传出 AWS 请求，我们对您的客户端进行了如下检测：

```
// Create a segment
ctx, root := xray.BeginSegment(context.TODO(), "AWSSDKV2_Dynamodb")
defer root.Close(nil)

cfg, err := config.LoadDefaultConfig(ctx, config.WithRegion("us-west-2"))
if err != nil {
    log.Fatalf("unable to load SDK config, %v", err)
}
// Instrumenting AWS SDK v2
awsv2.AWSV2Instrumentor(&cfg.APIOptions)
// Using the Config value, create the DynamoDB client
svc := dynamodb.NewFromConfig(cfg)
// Build the request with its input parameters
_, err = svc.ListTables(ctx, &dynamodb.ListTablesInput{
```

```
    Limit: aws.Int32(5),
})
if err != nil {
    log.Fatalf("failed to list tables, %v", err)
}
```

With OpenTelemetry SDK

的 AWS SDK for Go v2 Instrumen OpenTelemetry tion 提供了对下游 AWS SDK 调用的跟踪支持。以下是跟踪 S3 客户端调用的示例：

```
import (
    ...

    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/s3"

    "go.opentelemetry.io/otel"
    oteltrace "go.opentelemetry.io/otel/trace"
    awsConfig "github.com/aws/aws-sdk-go-v2/config"
    "go.opentelemetry.io/contrib/instrumentation/github.com/aws/aws-sdk-go-v2/
otelaws"
)

...


    // init aws config
    cfg, err := awsConfig.LoadDefaultConfig(ctx)
    if err != nil {
        panic("configuration error, " + err.Error())
    }

    // instrument all aws clients
    otelaws.AppendMiddlewares(&.APIOptions)


    // Call to S3
    s3Client := s3.NewFromConfig(cfg)
    input := &s3.ListBucketsInput{}
    result, err := s3Client.ListBuckets(ctx, input)
    if err != nil {
```

```
        fmt.Printf("Got an error retrieving buckets, %v", err)
        return
    }
```

检测传出 HTTP 调用

With X-Ray SDK

为了使用 X-Ray 检测传出的 HTTP 调用，Xray.Client 被用来创建所提供的 HTTP 客户端的副本。

```
myClient := xray.Client(http-client)

resp, err := ctxhttp.Get(ctx, xray.Client(nil), url)
```

With OpenTelemetry SDK

要使用 HTTP 客户端进行探测 OpenTelemetry，请使用 OpenTelemetry的 otelhttp。NewTransport 封装 http 的方法。 DefaultTransport。

```
import (
    "go.opentelemetry.io/contrib/instrumentation/net/http/otelhttp"
)

// Create an instrumented HTTP client.
httpClient := &http.Client{
    Transport: otelhttp.NewTransport(
        http.DefaultTransport,
    ),
}

req, err := http.NewRequestWithContext(ctx, http.MethodGet, "https://api.github.com/
repos/aws-observability/aws-otel-go/releases/latest", nil)
if err != nil {
    fmt.Printf("failed to create http request, %v\n", err)
}
res, err := httpClient.Do(req)
if err != nil {
    fmt.Printf("failed to make http request, %v\n", err)
}
```

```
// Request body must be closed
defer func(Body io.ReadCloser) {
    err := Body.Close()
    if err != nil {
        fmt.Printf("failed to close http response body, %v\n", err)
    }
}(res.Body)
```

对其他库的仪器支持

你可以在 Instrumentation 包下找 OpenTelemetry 到 Go 支持的库工具的完整列表。

或者，您可以在 OpenTelemetry 注册表中搜索注册表，以了解是否 OpenTelemetry 支持在注册表下为您的库提供工具。

手动创建跟踪数据

With X-Ray SDK

使用 X-Ray SDK，需要使用 BeginSegment 和 BeginSubsegment 方法来手动创建 X-Ray 分段和子分段。

```
// Start a segment
ctx, seg := xray.BeginSegment(context.Background(), "service-name")
// Start a subsegment
subCtx, subSeg := xray.BeginSubsegment(ctx, "subsegment-name")

// Add metadata or annotation here if necessary
xray.AddAnnotation(subCtx, "annotationKey", "annotationValue")
xray.AddMetadata(subCtx, "metadataKey", "metadataValue")

subSeg.Close(nil)
// Close the segment
seg.Close(nil)
```

With OpenTelemetry SDK

使用自定义跨度来监控未被仪器库捕获的内部活动的性能。请注意，只有服务器类型的跨度才会转换为 X-Ray 分段，所有其他跨度都会转换为 X-Ray 子分段。

首先，你需要创建一个 Tracer 来生成跨度，你可以通过该`otel.Tracer`方法获得跨度。这将提供在跟踪设置示例中全局注册 TracerProvider 的 Tracer 实例。您可以根据需要创建任意数量的 Tracer 实例，但通常为整个应用程序使用一个 Tracer。

```
    tracer := otel.Tracer("application-tracer")
```

```
import (
    ...

    oteltrace "go.opentelemetry.io/otel/trace"
)

...

    var attributes = []attribute.KeyValue{
        attribute.KeyValue{Key: "metadataKey", Value:
 attribute.StringValue("metadataValue")},
        attribute.KeyValue{Key: "annotationKey", Value:
 attribute.StringValue("annotationValue")},
        attribute.KeyValue{Key: "aws.xray.annotations", Value:
 attribute.StringSliceValue([]string{"annotationKey"})},
    }

    ctx := context.Background()

    parentSpanContext, parentSpan := tracer.Start(ctx,
 "ParentSpan", oteltrace.WithSpanKind(oteltrace.SpanKindServer),
 oteltrace.WithAttributes(attributes...))
    _, childSpan := tracer.Start(parentSpanContext, "ChildSpan",
 oteltrace.WithSpanKind(oteltrace.SpanKindInternal))

    // ...

    childSpan.End()
    parentSpan.End()
```

使用 OpenTelemetry SDK 向跟踪添加注释和元数据

在上面的示例中，该`WithAttributes`方法用于向每个跨度添加属性。请注意，默认情况下，所有 span 属性都将转换为 X-Ray 原始数据中的元数据。为确保将属性转换为注释而不是元数据，请将该属性的键添加到`aws.xray.annotations`属性列表中。有关更多信息，请参见启用自定义 X-Ray 注释。

Lambda 手动检测

With X-Ray SDK

使用 X-Ray SDK，在您的 Lambda 启用活动跟踪后，无需进行其他配置即可使用 X-Ray SDK。Lambda 创建了一个表示 Lambda 处理程序调用的区段，您使用 X-Ray SDK 创建了子分段，而无需进行任何其他配置。

With OpenTelemetry SDK

以下 Lambda 函数代码（不带工具）发出 Amazon S3 ListBuckets 调用和传出 HTTP 请求。

```
package main

import (
    "context"
    "encoding/json"
    "fmt"
    "io"
    "net/http"
    "os"

    "github.com/aws/aws-lambda-go/events"
    "github.com/aws/aws-lambda-go/lambda"
    awsconfig "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/s3"
)

func lambdaHandler(ctx context.Context) (interface{}, error) {
    // Initialize AWS config.
    cfg, err := awsconfig.LoadDefaultConfig(ctx)
    if err != nil {
        panic("configuration error, " + err.Error())
    }

    s3Client := s3.NewFromConfig(cfg)

    // Create an HTTP client.
    httpClient := &http.Client{
        Transport: http.DefaultTransport,
    }

    input := &s3.ListBucketsInput{}
```

```
    result, err := s3Client.ListBuckets(ctx, input)
    if err != nil {
        fmt.Printf("Got an error retrieving buckets, %v", err)
    }

    fmt.Println("Buckets:")
    for _, bucket := range result.Buckets {
        fmt.Println(*bucket.Name + ": " + bucket.CreationDate.Format("2006-01-02
 15:04:05 Monday"))
    }
    fmt.Println("End Buckets.")

    req, err := http.NewRequestWithContext(ctx, http.MethodGet, "https://
api.github.com/repos/aws-observability/aws-otel-go/releases/latest", nil)
    if err != nil {
        fmt.Printf("failed to create http request, %v\n", err)
    }
    res, err := httpClient.Do(req)
    if err != nil {
        fmt.Printf("failed to make http request, %v\n", err)
    }
    defer func(Body io.ReadCloser) {
        err := Body.Close()
        if err != nil {
            fmt.Printf("failed to close http response body, %v\n", err)
        }
    }(res.Body)

    var data map[string]interface{}
    err = json.NewDecoder(res.Body).Decode(&data)
    if err != nil {
        fmt.Printf("failed to read http response body, %v\n", err)
    }
    fmt.Printf("Latest ADOT Go Release is '%s'\n", data["name"])

    return events.APIGatewayProxyResponse{
        StatusCode: http.StatusOK,
        Body:       os.Getenv("_X_AMZN_TRACE_ID"),
    }, nil
}

func main() {
    lambda.Start(lambdaHandler)
}
```

要手动检测您的 Lambda 处理程序和 Amazon S3 客户端，请执行以下操作：

1. 在 main () 中，实例化 a TracerProvider (tp) 并将其注册为全局示踪器提供者。建议配置为：TracerProvider
   a. 带有 X-Ray UDP 跨度导出器的简单跨度处理器，可向 Lambda 的 UDP X-Ray 端点发送跟踪
   b. 将 service.name 设置为 Lambda 函数名称的资源
2. 将的用法更改`lambda.Start(lambdaHandler)`为`lambda.Start(otellambda.InstrumentHandler(lam`
   `xrayconfig.WithRecommendedOptions(tp)...))`。
3. 通过将 OpenTelemetry 中间件附加到 Amazon S3 客户端配置`aws-sdk-go-v2`中，使用 OpenTemetry AWS SDK 工具包检测 Amazon S3 客户端。
4. 使用 OpenTelemetry's `otelhttp.NewTransport` 方法封装 http 客户端`http.DefaultTransport`。

以下代码是一个示例，说明了更改后 Lambda 函数的外观。除了自动提供的跨度之外，您还可以手动创建其他自定义跨度。

```
package main

import (
    "context"
    "encoding/json"
    "fmt"
    "io"
    "net/http"
    "os"

    "github.com/aws-observability/aws-otel-go/exporters/xrayudp"
    "github.com/aws/aws-lambda-go/events"
    "github.com/aws/aws-lambda-go/lambda"
    awsconfig "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/s3"

    lambdadetector "go.opentelemetry.io/contrib/detectors/aws/lambda"
    "go.opentelemetry.io/contrib/instrumentation/github.com/aws/aws-lambda-go/
otellambda"
```

```
    "go.opentelemetry.io/contrib/instrumentation/github.com/aws/aws-lambda-go/
otellambda/xrayconfig"
    "go.opentelemetry.io/contrib/instrumentation/github.com/aws/aws-sdk-go-v2/
otelaws"
    "go.opentelemetry.io/contrib/instrumentation/net/http/otelhttp"
    "go.opentelemetry.io/contrib/propagators/aws/xray"
    "go.opentelemetry.io/otel"
    "go.opentelemetry.io/otel/attribute"
    "go.opentelemetry.io/otel/sdk/resource"
    "go.opentelemetry.io/otel/sdk/trace"
    semconv "go.opentelemetry.io/otel/semconv/v1.26.0"
)

func lambdaHandler(ctx context.Context) (interface{}, error) {
    // Initialize AWS config.
    cfg, err := awsconfig.LoadDefaultConfig(ctx)
    if err != nil {
        panic("configuration error, " + err.Error())
    }

    // Instrument all AWS clients.
    otelaws.AppendMiddlewares(&cfg.APIOptions)
    // Create an instrumented S3 client from the config.
    s3Client := s3.NewFromConfig(cfg)

    // Create an instrumented HTTP client.
    httpClient := &http.Client{
        Transport: otelhttp.NewTransport(
            http.DefaultTransport,
        ),
    }

    // return func(ctx context.Context) (interface{}, error) {
    input := &s3.ListBucketsInput{}
    result, err := s3Client.ListBuckets(ctx, input)
    if err != nil {
        fmt.Printf("Got an error retrieving buckets, %v", err)
    }

    fmt.Println("Buckets:")
    for _, bucket := range result.Buckets {
        fmt.Println(*bucket.Name + ": " + bucket.CreationDate.Format("2006-01-02
 15:04:05 Monday"))
    }
```

```
    fmt.Println("End Buckets.")

    req, err := http.NewRequestWithContext(ctx, http.MethodGet, "https://
api.github.com/repos/aws-observability/aws-otel-go/releases/latest", nil)
    if err != nil {
        fmt.Printf("failed to create http request, %v\n", err)
    }
    res, err := httpClient.Do(req)
    if err != nil {
        fmt.Printf("failed to make http request, %v\n", err)
    }
    defer func(Body io.ReadCloser) {
        err := Body.Close()
        if err != nil {
            fmt.Printf("failed to close http response body, %v\n", err)
        }
    }(res.Body)

    var data map[string]interface{}
    err = json.NewDecoder(res.Body).Decode(&data)
    if err != nil {
        fmt.Printf("failed to read http response body, %v\n", err)
    }
    fmt.Printf("Latest ADOT Go Release is '%s'\n", data["name"])

    return events.APIGatewayProxyResponse{
        StatusCode: http.StatusOK,
        Body:       os.Getenv("_X_AMZN_TRACE_ID"),
    }, nil
}

func main() {
    ctx := context.Background()
    detector := lambdadetector.NewResourceDetector()
    lambdaResource, err := detector.Detect(context.Background())
    if err != nil {
        fmt.Printf("failed to detect lambda resources: %v\n", err)
    }

    var attributes = []attribute.KeyValue{
        attribute.KeyValue{Key: semconv.ServiceNameKey, Value:
 attribute.StringValue(os.Getenv("AWS_LAMBDA_FUNCTION_NAME"))},
    }
    customResource := resource.NewWithAttributes(semconv.SchemaURL, attributes...)
```

```
    mergedResource, _ := resource.Merge(lambdaResource, customResource)

    xrayUdpExporter, _ := xrayudp.NewSpanExporter(ctx)
    tp := trace.NewTracerProvider(
        trace.WithSpanProcessor(trace.NewSimpleSpanProcessor(xrayUdpExporter)),
        trace.WithResource(mergedResource),
    )

    defer func(ctx context.Context) {
        err := tp.Shutdown(ctx)
        if err != nil {
            fmt.Printf("error shutting down tracer provider: %v", err)
        }
    }(ctx)

    otel.SetTracerProvider(tp)
    otel.SetTextMapPropagator(xray.Propagator{})

    lambda.Start(otellambda.InstrumentHandler(lambdaHandler,
 xrayconfig.WithRecommendedOptions(tp)...))
}
```

调用 Lambda 时，您将在控制台中Trace Map看到以下跟踪： CloudWatch

迁移到 OpenTelemetry Node.js

本节介绍如何将你的 Node.js 应用程序从 X-Ray SDK 迁移到 OpenTelemetry。它涵盖了自动和手动检测方法，并提供了常见用例的具体示例。

X-Ray Node.js SDK 可帮助你手动检测 Node.js 应用程序以进行跟踪。本节提供从 X-Ray 迁移到 OpenTelemetry 仪器的代码示例。

Sections

零代码自动检测解决方案

要使用适用于 Node.js 的 X-Ray SDK 跟踪请求，必须修改应用程序代码。借 OpenTelemetry助，您可以使用零代码自动检测解决方案来跟踪请求。

使用 OpenTelemetry基于自动仪器的零码自动检测。

1. 将 AWS 发行版用于 Node.js 的 OpenTelemetry (ADOT) 自动检测 — 有关 Node.js 应用程序的自动检测，请参阅使用AWS 发行版进行跟踪和衡量指标以进行自动插入。 OpenTelemetry JavaScript

   （可选）您还可以在使用 ADOT JavaScript 自动检测 CloudWatch 应用程序时启用应用程序信号，以监控当前应用程序的运行状况并根据 AWS 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。有关更多信息，请参阅 Application Signals。
2. 使用 OpenTelemetry JavaScript 零代码自动检测-要使用自动检测 OpenTelemetry JavaScript，请参阅JavaScript 零代码检测。

手动仪器解决方案

Tracing setup with X-Ray SDK

使用适用于 Node.js 的 X-Ray SDK 时，aws-xray-sdk软件包需要在使用 SDK 对您的代码进行检测之前，使用服务插件或本地采样规则配置 X-Ray SDK。

```
var AWSXRay = require('aws-xray-sdk');

AWSXRay.config([AWSXRay.plugins.EC2Plugin,AWSXRay.plugins.ElasticBeanstalkPlugin]);
AWSXRay.middleware.setSamplingRules(<path to file>);
```

Tracing setup with OpenTelemetry SDK

> **Note**
>
> AWS 目前无法为 OpenTelemetry JS 配置 X-Ray 远程采样。但是，目前可通过适用于 Node.js 的 ADOT Auto-Instrumenting 来支持 X-Ray 远程采样。

对于下面的代码示例，您将需要以下依赖项：

```
npm install --save \
  @opentelemetry/api \
  @opentelemetry/sdk-node \
  @opentelemetry/exporter-trace-otlp-proto \
  @opentelemetry/propagator-aws-xray \
  @opentelemetry/resource-detector-aws
```

在运行应用程序代码之前，必须设置和配置 OpenTelemetry SDK。这可以通过使用 —-req uire 标志来完成。创建一个名为 instrumentation.js 的文件，其中将包含您的 OpenTelemetry 仪器配置和设置。

建议您配置以下组件：

- OTLPTrace导出器-需要向 CloudWatch 代理/收集器导出追踪信息 OpenTelemetry
- AWSXRay传播器-需要将跟踪上下文传播到与 X-Ray 集成的 AWS 服务

- 资源检测器（例如，Amazon EC2 资源检测器）-检测运行您的应用程序的主机的元数据

```
/*instrumentation.js*/
// Require dependencies
const { NodeSDK } = require('@opentelemetry/sdk-node');
const { OTLPTraceExporter } = require('@opentelemetry/exporter-trace-otlp-proto');
const { AWSXRayPropagator } = require("@opentelemetry/propagator-aws-xray");
const { detectResources } = require('@opentelemetry/resources');
const { awsEc2Detector } = require('@opentelemetry/resource-detector-aws');

const resource = detectResources({
    detectors: [awsEc2Detector],
});

const _traceExporter = new OTLPTraceExporter({
    url: 'http://localhost:4318/v1/traces'
});

const sdk = new NodeSDK({
    resource: resource,
    textMapPropagator: new AWSXRayPropagator(),
    traceExporter: _traceExporter
});

sdk.start();
```

然后，你可以用你的 OpenTelemetry 设置来运行你的应用程序，比如：

```
node --require ./instrumentation.js app.js
```

您可以使用 OpenTelemetry SDK 库工具自动为 SDK 等库创建跨度。 AWS 启用这些功能将自动为模块（例如 JavaScript v3 版 AWS SDK）创建跨度。 OpenTelemetry 提供了启用所有库乐器或指定要启用哪些库乐器的选项。

要启用所有工具，请安装软件包：`@opentelemetry/auto-instrumentations-node`

```
npm install @opentelemetry/auto-instrumentations-node
```

接下来，更新配置以启用所有库工具，如下所示。

```
const { getNodeAutoInstrumentations } = require('@opentelemetry/auto-
instrumentations-node');

...

const sdk = new NodeSDK({
    resource: resource,
     instrumentations: [getNodeAutoInstrumentations()],
     textMapPropagator: new AWSXRayPropagator(),
    traceExporter: _traceExporter
});
```

Tracing setup with ADOT auto-instrumentation for Node.js

你可以使用适用于 Node.js 的 ADOT 自动检测来自动为你的 Node.js OpenTelemetry 应用程序进行配置。通过使用 ADOT Auto-Instrumention，您无需手动更改代码即可跟踪传入的请求或跟踪 AWS SDK 或 HTTP 客户端等库。有关更多信息，请参阅使用 OpenTelemetry JavaScript 自动检测 AWS 发行版进行跟踪和指标。

适用于 Node.js 的 ADOT 自动检测支持：

- 通过环境变量进行 X-Ray 远程采样 — `export OTEL_TRACES_SAMPLER=xray`
- X-Ray 轨迹上下文传播（默认启用）
- 资源检测（亚马逊 EC2、亚马逊 ECS 和 Amazon EKS 环境的资源检测默认处于启用状态）
- 对所有支持的 OpenTelemetry 仪器进行自动库插入，可以通过和环境变量来 disabled/enabled 选择性地进行库插入 `OTEL_NODE_ENABLED_INSTRUMENTATIONS` `OTEL_NODE_DISABLED_INSTRUMENTATIONS`
- 手动创建跨度

跟踪传入的请求

With X-Ray SDK

Express.js

使用 X-Ray SDK 来跟踪 Express.js 应用程序收到的传入 HTTP 请求，两个中间件`AWSXRay.express.openSegment(<name>)`和`AWSXRay.express.closeSegment()`需要封装所有定义的路由才能跟踪它们。

```
app.use(xrayExpress.openSegment('defaultName'));

...

app.use(xrayExpress.closeSegment());
```

Restify

为了跟踪`Restify`应用程序收到的传入 HTTP 请求，使用了 X-Ray SDK 中的中间件，方法是从 Restify `aws-xray-sdk-restify` 服务器上的模块运行启用：

```
var AWSXRay = require('aws-xray-sdk');
var AWSXRayRestify = require('aws-xray-sdk-restify');

var restify = require('restify');
var server = restify.createServer();
AWSXRayRestify.enable(server, 'MyApp'));
```

With OpenTelemetry SDK

Express.js

OpenTelemetry HTTP 检测和OpenTelemetry 快速检测`Express.js`为传入的请求提供跟踪支持。使用以下方法安装以下依赖项npm：

```
npm install --save @opentelemetry/instrumentation-http @opentelemetry/
instrumentation-express
```

更新 OpenTelemetry SDK 配置以启用对 express 模块的检测：

```
const { HttpInstrumentation } = require('@opentelemetry/instrumentation-http');
const { ExpressInstrumentation } = require('@opentelemetry/instrumentation-
express');
...

const sdk = new NodeSDK({
  ...

  instrumentations: [
    ...
    // Express instrumentation requires HTTP instrumentation
    new HttpInstrumentation(),
    new ExpressInstrumentation(),
  ],
});
```

Restify

对于 Restify 应用程序，你需要使用 OpenTelemetry Restify 工具。安装以下依赖项：

```
npm install --save @opentelemetry/instrumentation-restify
```

更新 OpenTelemetry SDK 配置以启用 restify 模块的检测功能：

```
const { RestifyInstrumentation } = require('@opentelemetry/instrumentation-
restify');
...

const sdk = new NodeSDK({
  ...

  instrumentations: [
    ...
    new RestifyInstrumentation(),
  ],
});
```

AWS SDK JavaScript V3 插件

With X-Ray SDK

要检测来自 AWS SDK 的传出 AWS 请求，您需要对客户端进行检测，如下例所示：

```
import { S3, PutObjectCommand } from '@aws-sdk/client-s3';

const s3 = AWSXRay.captureAWSv3Client(new S3({}));

await s3.send(new PutObjectCommand({
  Bucket: bucketName,
  Key: keyName,
  Body: 'Hello!',
}));
```

With OpenTelemetry SDK

对下游 AWS DynamoDB、Amazon S3 和其他软件开发工具包调用的跟踪支持由 OpenTelemetry AWS 软件开发工具包工具包工具提供。使用以下方法安装以下依赖项npm：

```
npm install --save @opentelemetry/instrumentation-aws-sdk
```

使用 S OpenTelemetry DK 工具更新 S AWS DK 配置。

```
import { AwsInstrumentation } from '@opentelemetry/instrumentation-aws-sdk';
...

const sdk = new NodeSDK({
  ...

  instrumentations: [
    ...
    new AwsInstrumentation()
  ],
});
```

检测传出 HTTP 调用

With X-Ray SDK

要使用 X-Ray 检测传出的 HTTP 请求，需要检测客户端。例如，见下文。

单个 HTTP 客户端

```
var AWSXRay = require('aws-xray-sdk');
var http = AWSXRay.captureHTTPs(require('http'));
```

所有 HTTP 客户端（全局）

```
var AWSXRay = require('aws-xray-sdk');
AWSXRay.captureHTTPsGlobal(require('http'));
var http = require('http');
```

With OpenTelemetry SDK

HTTP 工具为 Node.js HTT OpenTelemetry P 客户端提供跟踪支持。使用以下方法安装以下依赖项npm：

```
npm install --save @opentelemetry/instrumentation-http
```

按如下方式更新 OpenTelemetry SDK 配置：

```
const { HttpInstrumentation } = require('@opentelemetry/instrumentation-http');
...

const sdk = new NodeSDK({
  ...

  instrumentations: [
    ...
    new HttpInstrumentation(),
```

```
  ],
});
```

对其他库的仪器支持

[您可以在 “支持的仪器” OpenTelemetry JavaScript 下找到支持的库工具的完整列表。]

或者，您可以在 OpenTelemetry 注册表中搜索注册表，以了解是否 OpenTelemetry 支持在注册表下为您的库提供工具。

手动创建跟踪数据

With X-Ray SDK

使用 X-Ray，需要使用`aws-xray-sdk`软件包代码来手动创建区段及其子子分段来跟踪您的应用程序。

```
var AWSXRay = require('aws-xray-sdk');

AWSXRay.enableManualMode();

var segment = new AWSXRay.Segment('myApplication');

captureFunc('1', function(subsegment1) {
  captureFunc('2', function(subsegment2) {

  }, subsegment1);
}, segment);

segment.close();
segment.flush();
```

With OpenTelemetry SDK

您可以创建和使用自定义跨度来监控未被仪器库捕获的内部活动的性能。请注意，只有服务器类型的跨度才会转换为 X-Ray 分段，所有其他跨度都会转换为 X-Ray 子分段。有关更多信息，请参阅分段。

在跟踪设置中配置 OpenTelemetry SDK 以创建 Span 之后，您将需要一个 Tracer 实例。您可以根据需要创建任意数量的 Tracer 实例，但通常为整个应用程序使用一个 Tracer。

```
const { trace, SpanKind } = require('@opentelemetry/api');

// Get a tracer instance
const tracer = trace.getTracer('your-tracer-name');

...

  // This span will appear as a segment in X-Ray
  tracer.startActiveSpan('server', { kind: SpanKind.SERVER }, span => {
    // Do work here

    // This span will appear as a subsegment in X-Ray
    tracer.startActiveSpan('operation2', { kind: SpanKind.INTERNAL }, innerSpan => {
      // Do more work here

      innerSpan.end();
    });
    span.end();
  });
```

使用 OpenTelemetry SDK 向跟踪添加注释和元数据

您还可以将自定义键值对添加为跨度中的属性。请注意，默认情况下，所有这些跨度属性都将转换为 X-Ray 原始数据中的元数据。为确保将属性转换为注释而不是元数据，请将该属性的键添加到`aws.xray.annotations`属性列表中。有关更多信息，请参见启用自定义 X-Ray 注释。

```
 tracer.startActiveSpan('server', { kind: SpanKind.SERVER }, span => {
    span.setAttribute('metadataKey', 'metadataValue');
    span.setAttribute('annotationKey', 'annotationValue');

    // The following ensures that "annotationKey: annotationValue" is an annotation
 in X-Ray raw data.
    span.setAttribute('aws.xray.annotations', ['annotationKey']);

    // Do work here

    span.end();
```

```
});
```

Lambda 仪器

With X-Ray SDK

为 Lambda 函数启用主动跟踪后，无需额外配置即可使用 X-Ray SDK。Lambda 创建了一个表示 Lambda 处理程序调用的区段，而您使用 X-Ray SDK 创建了子分段或仪器库，而无需进行任何其他配置。

With OpenTelemetry SDK

您可以使用已打开的 Lambda 图层自动检测您的 Lambda AWS 。有两种解决方案：

- （推荐）CloudWatch 应用程序信号 lambda 层

> **Note**
>
> 此 Lambda 层默认启用 CloudWatch 应用程序信号，通过收集指标和跟踪来监控您的 Lambda 应用程序的性能和运行状况。如果您只想要跟踪，则应设置 Lambda 环境变量。`OTEL_AWS_APPLICATION_SIGNALS_ENABLED=false`有关更多信息，请参阅在 Lambda 上启用您的应用程序。

- AWS 适用于 ADOT JS 的托管 Lambda 层。有关更多信息，请参阅适用于 Lamb AWS d OpenTelemetry a Support 的发行版。 JavaScript

使用 Lambda 工具手动创建跨度

虽然 ADOT Lamb JavaScript da 层为您的 Lambda 函数提供了自动检测，但您可能会发现需要在 Lambda 中执行手动检测，例如，提供自定义数据或在 Lambda 函数本身中检测库工具未涵盖的代码。

要在自动检测的同时执行手动插入，您需要添加`@opentelemetry/api`为依赖项。建议此依赖项的版本与 ADOT JavaScript SDK 使用的相同依赖项的版本相同。您可以使用 OpenTelemetry API 在 Lambda 函数中手动创建跨度。

要使用 NPM 添加`@opentelemetry/api`依赖项，请执行以下操作：

```
npm install @opentelemetry/api
```

迁移到 OpenTelemetry .NET

在.NET 应用程序中使用 X-Ray Tracing 时，使用带有手动操作的 X-Ray .NET SDK 进行检测。

本节在使用 SDK 的手动仪器解决方案部分中提供了从 X-Ray 手动检测解决方案迁移到.NET 的 OpenTelemetry 手动检测解决方案的代码示例。或者，您可以从 X-Ray 手动检测迁移到 OpenTelemetry 自动检测解决方案，以检验.NET 应用程序，而不必零代码自动检测解决方案在本节中修改应用程序源代码。

Sections

零代码自动检测解决方案

OpenTelemetry 提供零代码自动检测解决方案。这些解决方案无需更改应用程序代码即可跟踪请求。

OpenTelemetry基于自动仪表选项

1. 使用适用于.NET 的 OpenTelemetry (ADOT) 自动检测 AWS 发行版 — 要自动检测.NET 应用程序，请参阅使用适用于.NET 自动检测的AWS 发行版进行跟踪和衡量指标。 OpenTelemetry

   （可选）在 CloudWatch 使用 ADOT .NET 自动检测应用程序时启用应用程序信号， AWS 以：

   - 监控当前应用程序的运行状况
   - 根据业务目标跟踪长期应用程序性能
   - 获得以应用程序为中心的统一视图，了解您的应用程序、服务和依赖关系

- 监控和分类应用程序运行状况

有关更多信息，请参阅 Application Signals。

2. 使用 OpenTelemetry .Net 零代码自动检测 — 要自动使用 OpenTelemetry .NET 进行检测，请参阅使用.NE OpenTelemetry T 自动检测 AWS 发行版进行跟踪和衡量指标。

使用 SDK 的手动仪器解决方案

Tracing configuration with X-Ray SDK

对于.NET Web 应用程序，X-Ray SDK 是在`Web.config`文件的 AppSettings 部分配置的。

网页配置示例

```
<configuration>
  <appSettings>
    <add key="AWSXRayPlugins" value="EC2Plugin"/>
  </appSettings>
</configuration>
```

对于.NET Core，使用名`appsettings.json`XRay为的顶级密钥命名的文件，然后构建一个配置对象来初始化 X-Ray 记录器。

.NET 的示例 `appsettings.json`

```
{
  "XRay": {
    "AWSXRayPlugins": "EC2Plugin"
  }
}
```

.NET Core Program.cs 示例 — 录制器配置

```
using Amazon.XRay.Recorder.Core;
...
```

```
AWSXRayRecorder.InitializeInstance(configuration);
```

Tracing configuration with OpenTelemetry SDK

添加以下依赖项：

```
dotnet add package OpenTelemetry
dotnet add package OpenTelemetry.Contrib.Extensions.AWSXRay
dotnet add package OpenTelemetry.Sampler.AWS --prerelease
dotnet add package OpenTelemetry.Resources.AWS
dotnet add package OpenTelemetry.Exporter.OpenTelemetryProtocol
dotnet add package OpenTelemetry.Extensions.Hosting
dotnet add package OpenTelemetry.Instrumentation.AspNetCore
```

对于您的.NET 应用程序，请通过设置全局来配置 OpenTelemetry SDK TracerProvider。以下示例配置还为启用了检测功能ASP.NET Core。要进行仪器ASP.NET，请参阅跟踪传入的请求（ASP.NET 和 ASP.NET 核心工具）。要与其他框架 OpenTelemetry 一起使用，请参阅 Regist ry 以获取更多支持的框架的库。

建议您配置以下组件：

- An OTLP Exporter— 将跟踪导出到 CloudWatch Agent/Collecto OpenTelemetry r 时需要此选项
- AWS X-Ray 传播器 — 将跟踪上下文传播到与 X-Ray 集成的AWS 服务所必需的
- AWS X-Ray-Remote 采样器 — 如果您需要使用 X-Ray 采样规则对请求进行采样，则为必填项
- Resource Detectors（例如，Amazon EC2 资源检测器）-检测运行您的应用程序的主机的元数据

```
using OpenTelemetry;
using OpenTelemetry.Contrib.Extensions.AWSXRay.Trace;
using OpenTelemetry.Sampler.AWS;
using OpenTelemetry.Trace;
using OpenTelemetry.Resources;

var builder = WebApplication.CreateBuilder(args);
```

```
var serviceName = "MyServiceName";
var serviceVersion = "1.0.0";

var resourceBuilder = ResourceBuilder
    .CreateDefault()
    .AddService(serviceName: serviceName)
    .AddAWSEC2Detector();

builder.Services.AddOpenTelemetry()
    .ConfigureResource(resource => resource
        .AddAWSEC2Detector()
        .AddService(
            serviceName: serviceName,
            serviceVersion: serviceVersion))
    .WithTracing(tracing => tracing
        .AddSource(serviceName)
        .AddAspNetCoreInstrumentation()
        .AddOtlpExporter()
        .SetSampler(AWSXRayRemoteSampler.Builder(resourceBuilder.Build())
            .SetEndpoint("http://localhost:2000")
            .Build()));

Sdk.SetDefaultTextMapPropagator(new AWSXRayPropagator()); // configure  X-Ray
 propagator
```

要 OpenTelemetry 用于控制台应用程序，请在程序启动时添加以下 OpenTelemetry 配置。

```
using OpenTelemetry;
using OpenTelemetry.Contrib.Extensions.AWSXRay.Trace;
using OpenTelemetry.Trace;
using OpenTelemetry.Resources;

var serviceName = "MyServiceName";

var resourceBuilder = ResourceBuilder
    .CreateDefault()
    .AddService(serviceName: serviceName)
    .AddAWSEC2Detector();

var tracerProvider = Sdk.CreateTracerProviderBuilder()
```

```
    .AddSource(serviceName)
    .ConfigureResource(resource =>
        resource
            .AddAWSEC2Detector()
            .AddService(
                serviceName: serviceName,
                serviceVersion: serviceVersion
            )
        )
    .AddOtlpExporter() // default address localhost:4317
    .SetSampler(new TraceIdRatioBasedSampler(1.00))
    .Build();

Sdk.SetDefaultTextMapPropagator(new AWSXRayPropagator()); // configure  X-Ray
 propagator
```

手动创建跟踪数据

With X-Ray SDK

使用 X-Ray SDK，需要使用`BeginSegment`和`BeginSubsegment`方法来手动创建 X-Ray 分段和子分段。

```
using Amazon.XRay.Recorder.Core;

AWSXRayRecorder.Instance.BeginSegment("segment name"); // generates `TraceId` for
 you
try
{
    // Do something here
    // can create custom subsegments
    AWSXRayRecorder.Instance.BeginSubsegment("subsegment name");
    try
    {
        DoSometing();
    }
    catch (Exception e)
    {
        AWSXRayRecorder.Instance.AddException(e);
    }
```

```
    finally
    {
        AWSXRayRecorder.Instance.EndSubsegment();
    }
}
catch (Exception e)
{
    AWSXRayRecorder.Instance.AddException(e);
}
finally
{
    AWSXRayRecorder.Instance.EndSegment();
}
```

With OpenTelemetry SDK

在.NET 中，您可以使用活动 API 创建自定义跨度，以监控未被插桩库捕获的内部活动的性能。请注意，只有服务器类型的跨度才会转换为 X-Ray 分段，所有其他跨度都会转换为 X-Ray 子段。

您可以根据需要创建任意数量的`ActivitySource`实例，但建议整个应用程序/服务只创建一个实例。

```
using System.Diagnostics;

ActivitySource activitySource = new ActivitySource("ActivitySourceName",
 "ActivitySourceVersion");


...


using (var activity = activitySource.StartActivity("ActivityName",
 ActivityKind.Server)) // this will be translated to a X-Ray Segment
{
    // Do something here

    using (var internalActivity = activitySource.StartActivity("ActivityName",
 ActivityKind.Internal)) // this will be translated to an X-Ray Subsegment
    {
        // Do something here
    }
```

```
}
```

使用 OpenTelemetry SDK 向跟踪添加注释和元数据

您还可以通过在活动中使用SetTag方法将自定义键值对作为属性添加到跨度上。请注意，默认情况下，所有 span 属性都将转换为 X-Ray 原始数据中的元数据。为确保将属性转换为注释而不是元数据，您可以将该属性的密钥添加到aws.xray.annotations属性列表中。

```
using (var activity = activitySource.StartActivity("ActivityName",
 ActivityKind.Server)) // this will be translated to a X-Ray Segment
{
    activity.SetTag("metadataKey", "metadataValue");
    activity.SetTag("annotationKey", "annotationValue");
    string[] annotationKeys = {"annotationKey"};
    activity.SetTag("aws.xray.annotations", annotationKeys);

    // Do something here

    using (var internalActivity = activitySource.StartActivity("ActivityName",
 ActivityKind.Internal)) // this will be translated to an X-Ray Subsegment
    {
        // Do something here
    }
}
```

使用 OpenTelemetry 自动仪器

如果您使用的是适用于.NET 的 OpenTelemetry 自动插桩解决方案，并且需要在应用程序中执行手动插入，例如，在应用程序本身中检测任何自动插桩库未涵盖的部分的代码。

由于只能有一个全局插入TracerProvider，因此TracerProvider如果与自动检测一起使用，则手动插桩不应自行实例化。使用TracerProvider时，自定义手动跟踪的工作方式与通过 OpenTelemetry SDK 使用自动检测或手动检测的方式相同。

跟踪传入的请求（ASP.NET 和 ASP.NET 核心工具）

With X-Ray SDK

要检测 ASP.NET 应用程序提供的请求，https://docs.aws.amazon.com/xray/latest/devguide/xray-sdk-dotnet-messagehandler.html请参阅，了解有关如何调用`RegisterXRay``global.asax`文件`Init`方法的信息。

```
AWSXRayASPNET.RegisterXRay(this, "MyApp");
```

要检测由 ASP.NET 核心应用程序处理的请求，则在启动类的`UseXRay`方法中的任何其他中间件之前调`Configure`用该方法。

```
app.UseXRay("MyApp");
```

With OpenTelemetry SDK

OpenTelemetry 还提供了用于收集 ASP.NET 和 ASP.NET 内核传入 Web 请求的跟踪的工具库。以下部分列出了为您的 OpenTelemetry 配置添加和启用这些库工具所需的步骤，包括在创建 Tracer Provider 时如何添加 ASP.NET 或 ASP.NET 核心工具。

有关如何启用 OpenTelemetry .Instrumention 的信息。 AspNet，请参阅启用 OpenTelemetry .Instrumention 的步骤。 AspNet以及有关如何启用 OpenTelemetry .Instrumention 的信息。 AspNetCore，请参阅启用 OpenTelemetry .Instrumention 的步骤。 AspNetCore。

AWS 软件开发工具包工具

With X-Ray SDK

通过调用安装所有 AWS SDK 客户端`RegisterXRayForAllServices()`。

```
using Amazon.XRay.Recorder.Handlers.AwsSdk;
AWSSDKHandler.RegisterXRayForAllServices(); //place this before any instantiation of
 AmazonServiceClient
```

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient(RegionEndpoint.USWest2); //
 AmazonDynamoDBClient is automatically registered with X-Ray
```

使用以下方法之一进行特定的 AWS 服务客户端检测。

```
AWSSDKHandler.RegisterXRay<IAmazonDynamoDB>(); // Registers specific type of
 AmazonServiceClient : All instances of IAmazonDynamoDB created after this line are
 registered
AWSSDKHandler.RegisterXRayManifest(String path); // To configure custom AWS Service
 Manifest file. This is optional, if you have followed "Configuration" section
```

With OpenTelemetry SDK

对于以下代码示例，您将需要以下依赖关系：

```
dotnet add package OpenTelemetry.Instrumentation.AWS
```

要对 S AWS DK 进行检测，请更新设置全局 TracerProvider 的 OpenTelemetry SDK 配置。

```
builder.Services.AddOpenTelemetry()
    ...
    .WithTracing(tracing => tracing
        .AddAWSInstrumentation()
        ...
```

检测传出 HTTP 调用

With X-Ray SDK

X-Ray .NET SDK 通过扩展方法`GetResponseTraced()`或在使用时跟踪传出的 HTTP 调用`System.Net.HttpWebRequest`，或者`GetAsyncResponseTraced()`在使用时使用`HttpClientXRayTracingHandler`处理程序来跟踪传出的 HTTP 调用`System.Net.Http.HttpClient`。

With OpenTelemetry SDK

对于以下代码示例，您将需要以下依赖关系：

```
dotnet add package OpenTelemetry.Instrumentation.Http
```

要检测System.Net.Http.HttpClient和System.Net.HttpWebRequest，请更新设置全局 TracerProvider 的 OpenTelemetry SDK 配置。

```
builder.Services.AddOpenTelemetry()
    ...
    .WithTracing(tracing => tracing
        .AddHttpClientInstrumentation()
        ...
```

对其他库的仪器支持

您可以在 OpenTelemetry 注册表中搜索和筛选.NET 工具库，以了解您的库是否 OpenTelemetry 支持插入。请查看注册表开始搜索。

Lambda 仪器

With X-Ray SDK

要将 X-Ray 软件开发工具包与 Lambda 一起使用，需要执行以下步骤：

1. 在 Lambda 函数上启用主动跟踪
2. Lambda 服务会创建一个表示处理程序调用的分段
3. 使用 X-Ray SDK 创建子分段或仪器库

With OpenTelemetry-based solutions

您可以使用已打开的 Lambda 图层自动检测您的 Lambda AWS 。有两种解决方案：

- （推荐）CloudWatch 应用程序信号 lambda 层

- 为了提高性能，您可能需要考虑使用OpenTelemetry Manual Instrumentation为您的 Lambda 函数生成 OpenTelemetry 跟踪。

OpenTelemetry 针对 AWS Lambda 的手动检测

以下是 Lambda 函数代码（不带工具）示例。

```
using System;
using System.Text;
using System.Threading.Tasks;
using Amazon.Lambda.Core;
using Amazon.S3;
using Amazon.S3.Model;

// Assembly attribute to enable Lambda function logging
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer

namespace ExampleLambda;

public class ListBucketsHandler
{
    private static readonly AmazonS3Client s3Client = new();

    // new Lambda function handler passed in
    public async Task<string> HandleRequest(object input, ILambdaContext context)
    {
        try
        {
            var DoListBucketsAsyncResponse = await DoListBucketsAsync();
            context.Logger.LogInformation($"Results:
 {DoListBucketsAsyncResponse.Buckets}");

            context.Logger.LogInformation($"Successfully called ListBucketsAsync");
            return "Success!";
        }
        catch (Exception ex)
        {
            context.Logger.LogError($"Failed to call ListBucketsAsync: {ex.Message}");
            throw;
        }
    }
```

```
    private async Task<ListBucketsResponse> DoListBucketsAsync()
    {
        try
        {
            var putRequest = new ListBucketsRequest
            {
            };

            var response = await s3Client.ListBucketsAsync(putRequest);
            return response;
        }
        catch (AmazonS3Exception ex)
        {
            throw new Exception($"Failed to call ListBucketsAsync: {ex.Message}", ex);
        }
    }
}
```

要手动检测您的 Lambda 处理程序和 Amazon S3 客户端，请执行以下操作。

1. 实例化 a TracerProvider — 建议配置 TracerProvider 为 Always On Sampler，`service.name`设置为 Lambda 函数名称。`XrayUdpSpanExporter` ParentBased `Resource`
2. 通过调用将 S OpenTemetry AWS DK 客户端插桩添加到 Amazon S3 客户端，使用软件开发工具 AWS 包插`AddAWSInstrumentation()`件对 Amazon S3 客户端进行检测 `TracerProvider`
3. 创建一个与原始 Lambda 函数具有相同签名的包装函数。调用 `AWSLambdaWrapper.Trace()` API 并传递`TracerProvider`原始 Lambda 函数及其输入作为参数。将包装函数设置为 Lambda 处理程序的输入。

对于以下代码示例，您将需要以下依赖项：

```
dotnet add package OpenTelemetry.Instrumentation.AWSLambda
dotnet add package OpenTelemetry.Instrumentation.AWS
dotnet add package OpenTelemetry.Resources.AWS
dotnet add package AWS.Distro.OpenTelemetry.Exporter.Xray.Udp
```

以下代码演示了进行必要更改后的 Lambda 函数。您可以创建其他自定义跨度来补充自动提供的跨度。

```
using Amazon.Lambda.Core;
using Amazon.S3;
using Amazon.S3.Model;
using OpenTelemetry;
using OpenTelemetry.Instrumentation.AWSLambda;
using OpenTelemetry.Trace;
using AWS.Distro.OpenTelemetry.Exporter.Xray.Udp;
using OpenTelemetry.Resources;

// Assembly attribute to enable Lambda function logging
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer

namespace ExampleLambda;

public class ListBucketsHandler
{
    private static readonly AmazonS3Client s3Client = new();

    TracerProvider tracerProvider = Sdk.CreateTracerProviderBuilder()
        .AddAWSLambdaConfigurations()
        .AddProcessor(
            new SimpleActivityExportProcessor(
                // AWS_LAMBDA_FUNCTION_NAME Environment Variable will be defined in AWS
 Lambda Environment
                new
 XrayUdpExporter(ResourceBuilder.CreateDefault().AddService(Environment.GetEnvironmentVariable(
            )
        )
        .AddAWSInstrumentation()
        .SetSampler(new ParentBasedSampler(new AlwaysOnSampler()))
        .Build();

    // new Lambda function handler passed in
    public async Task<string> HandleRequest(object input, ILambdaContext context)
    => await AWSLambdaWrapper.Trace(tracerProvider, OriginalHandleRequest, input,
 context);
```

```
    public async Task<string> OriginalHandleRequest(object input, ILambdaContext
 context)
    {
        try
        {
            var DoListBucketsAsyncResponse = await DoListBucketsAsync();
            context.Logger.LogInformation($"Results:
 {DoListBucketsAsyncResponse.Buckets}");

            context.Logger.LogInformation($"Successfully called ListBucketsAsync");
            return "Success!";
        }
        catch (Exception ex)
        {
            context.Logger.LogError($"Failed to call ListBucketsAsync: {ex.Message}");
            throw;
        }
    }

    private async Task<ListBucketsResponse> DoListBucketsAsync()
    {
        try
        {
            var putRequest = new ListBucketsRequest
            {
            };

            var response = await s3Client.ListBucketsAsync(putRequest);
            return response;
        }
        catch (AmazonS3Exception ex)
        {
            throw new Exception($"Failed to call ListBucketsAsync: {ex.Message}", ex);
        }
    }
}
```

调用此 Lambda 时，您将在控制台的跟踪映射中 CloudWatch 看到以下跟踪：

迁移到 OpenTelemetry Python

本指南可帮助你将 Python 应用程序从 X-Ray SDK 迁移到 OpenTelemetry 仪器中。它涵盖了自动和手动检测方法，并提供了常见场景的代码示例。

Sections

零代码自动检测解决方案

使用 X-Ray SDK，您必须修改应用程序代码才能跟踪请求。 OpenTelemetry 提供用于跟踪请求的零代码自动检测解决方案。使用 OpenTelemetry，您可以选择使用零代码自动检测解决方案来跟踪请求。

使用 OpenTelemetry基于自动仪器的零代码

1. 将 AWS 发行版用于 Python 的 OpenTelemetry (ADOT) 自动插入 — 有关 Python 应用程序的自动检测，请参阅使用 Python 自动检测AWS 发行版进行 OpenTelemetry 跟踪和衡量指标。

   （可选）您还可以在使用 ADOT Python 自动检测应用程序时启用 CloudWatch 应用程序信号，以监控当前应用程序运行状况并根据 AWS 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。

2. 使用 OpenTelemetry Python 零代码自动检测 — 要使用 Py OpenTelemetry thon 进行自动检测，请参阅 Pyt hon 零代码检测。

手动检测您的应用程序

您可以使用pip命令手动检测应用程序。

With X-Ray SDK

```
pip install aws-xray-sdk
```

With OpenTelemetry SDK

```
pip install opentelemetry-api opentelemetry-sdk opentelemetry-exporter-otlp
 opentelemetry-propagator-aws-xray
```

跟踪设置初始化

With X-Ray SDK

在 X-Ray 中，全局xray_recorder被初始化并使用它来生成分段和子分段。

With OpenTelemetry SDK

> **Note**
>
> 目前无法为 OpenTelemetry Python 配置 X-Ray 远程采样。但是，目前可通过适用于 Python 的 ADOT 自动检测来支持 X-Ray 远程采样。

在中 OpenTelemetry，你需要初始化一个全局变量TracerProvider。使用它TracerProvider，你可以获得一个 Tracer，你可以用它在应用程序中的任何地方生成跨度。建议您配置以下组件：

- `OTLPSpanExporter`— 将跟踪导出到 CloudWatch Agent/Collecto OpenTelemetry r 时需要此选项
- AWS X-Ray 传播器 — 将跟踪上下文传播到与 X-Ray 集成的 AWS 服务所必需的

```
from opentelemetry import (
    trace,
    propagate
)
from opentelemetry.sdk.trace import TracerProvider

from opentelemetry import trace
from opentelemetry.sdk.trace import TracerProvider
from opentelemetry.exporter.otlp.proto.http.trace_exporter import OTLPSpanExporter
from opentelemetry.sdk.trace.export import BatchSpanProcessor
from opentelemetry.propagators.aws import AwsXRayPropagator

# Sends generated traces in the OTLP format to an OTel Collector running on port
 4318
otlp_exporter = OTLPSpanExporter(endpoint="http://localhost:4318/v1/traces")
# Processes traces in batches as opposed to immediately one after the other
span_processor = BatchSpanProcessor(otlp_exporter)
# More configurations can be done here. We will visit them later.

# Sets the global default tracer provider
provider = TracerProvider(active_span_processor=span_processor)
trace.set_tracer_provider(provider)

# Configures the global propagator to use the X-Ray Propagator
propagate.set_global_textmap(AwsXRayPropagator())

# Creates a tracer from the global tracer provider
tracer = trace.get_tracer("my.tracer.name")
# Use this tracer to create Spans
```

使用适用于 Python 的 ADOT 自动检测功能

你可以使用适用于 Python 的 ADOT 自动插桩来自动为你的 Python OpenTelemetry 应用程序进行配置。通过使用 ADOT 自动检测，您无需手动更改代码即可跟踪传入的请求或跟踪 AWS SDK 或 HTTP

客户端等库。有关更多信息，请参阅使用适用于 OpenTelemetry Python 自动检测的 AWS 发行版进行跟踪和指标。

适用于 Python 的 ADOT 自动检测支持：

- 通过环境变量进行 X-Ray 远程采样 `export OTEL_TRACES_SAMPLER=xray`
- X-Ray 轨迹上下文传播（默认启用）
- 资源检测（亚马逊 EC2、亚马逊 ECS 和 Amazon EKS 环境的资源检测默认处于启用状态）
- 默认情况下，所有支持的乐器的自动库 OpenTelemetry 乐器均处于启用状态。您可以通过`OTEL_PYTHON_DISABLED_INSTRUMENTATIONS` 环境变量有选择地禁用。（默认情况下全部启用）
- 手动创建跨度

从 X-Ray 服务插件到 OpenTelemetry AWS 资源提供商

X-Ray SDK 提供了插件，您可以将其添加到中，以从亚马逊 EC2、亚马逊 ECS 和 Elastic Beanstalk 等托管服务中捕获平台特定信息。`xray_recorder`它与中的资源提供者类似 OpenTelemetry ，它将信息捕获为资源属性。有多个资源提供器可用于不同的 AWS 平台。

- 首先安装 AWS 扩展包，`pip install opentelemetry-sdk-extension-aws`
- 配置所需的资源检测器。以下示例说明如何在 OpenTelemetry 软件开发工具包中配置 Amazon EC2 资源提供商

```
from opentelemetry import trace
from opentelemetry.sdk.trace import TracerProvider
from opentelemetry.sdk.extension.aws.resource.ec2 import (
    AwsEc2ResourceDetector,
)
from opentelemetry.sdk.resources import get_aggregated_resources

provider = TracerProvider(
    active_span_processor=span_processor,
    resource=get_aggregated_resources([
        AwsEc2ResourceDetector(),
    ]))

trace.set_tracer_provider(provider)
```

跟踪传入的请求

With X-Ray SDK

X-Ray Python SDK 支持 Django、Flask 和 Bottle 等应用程序框架来跟踪在其上运行的 Python 应用程序的传入请求。这是通过将每个框架`XRayMiddleware`添加到应用程序中来完成的。

With OpenTelemetry SDK

OpenTelemetry 通过特定的仪器库为 Django 和 Flask 提供工具。中没有适用于 Bottle 的工具 OpenTelemetry，仍然可以使用 OpenTelemetry WSGI Instrumention 来跟踪应用程序。

对于以下代码示例，您需要以下依赖关系：

```
pip install opentelemetry-instrumentation-flask
```

在为应用程序框架添加工具 TracerProvider 之前，必须初始化 OpenTelemetry SDK 并注册全局版本。没有它，追踪操作将是`no-ops`。配置完全局配置后`TracerProvider`，就可以将 instrumentor 用于您的应用程序框架。以下示例演示了一个 Flask 应用程序。

```
from flask import Flask
from opentelemetry import trace
from opentelemetry.instrumentation.flask import FlaskInstrumentor
from opentelemetry.sdk.extension.aws.resource import AwsEc2ResourceDetector
from opentelemetry.sdk.resources import get_aggregated_resources
from opentelemetry.sdk.trace import TracerProvider
from opentelemetry.sdk.trace.export import BatchSpanProcessor, ConsoleSpanExporter

provider = TracerProvider(resource=get_aggregated_resources(
    [
        AwsEc2ResourceDetector(),
    ]))

processor = BatchSpanProcessor(ConsoleSpanExporter())
provider.add_span_processor(processor)
trace.set_tracer_provider(provider)

# Creates a tracer from the global tracer provider
tracer = trace.get_tracer("my.tracer.name")
```

```
app = Flask(__name__)

# Instrument the Flask app
FlaskInstrumentor().instrument_app(app)


@app.route('/')
def hello_world():
    return 'Hello World!'


if __name__ == '__main__':
    app.run()
```

AWS 软件开发工具包工具

With X-Ray SDK

X-Ray Python AWS SDK 通过修补botocore库来跟踪 SDK 客户端请求。有关更多信息，请参阅使用适用于 Python 的 X-Ray SD AWS K 追踪 SDK 调用。在您的应用程序中，该patch_all()方法用于使用或库来检测所有库或有选择地botocore进行修补。boto3 patch(['botocore'])任何选定的方法都会检测应用程序中的所有 Boto3 客户端，并为使用这些客户端进行的任何调用生成子分段。

With OpenTelemetry SDK

对于以下代码示例，您将需要以下依赖关系：

```
pip install opentelemetry-instrumentation-botocore
```

以编程方式使用 OpenTelemetry Botocore Instrumenting 来检测应用程序中的所有 Boto3 客户端。以下示例演示了该botocore工具。

```
import boto3
import opentelemetry.trace as trace
from botocore.exceptions import ClientError
from opentelemetry.sdk.trace import TracerProvider
```

```
from opentelemetry.sdk.resources import get_aggregated_resources
from opentelemetry.sdk.trace.export import (
    BatchSpanProcessor,
    ConsoleSpanExporter,
)
from opentelemetry.instrumentation.botocore import BotocoreInstrumentor

provider = TracerProvider()
processor = BatchSpanProcessor(ConsoleSpanExporter())
provider.add_span_processor(processor)
trace.set_tracer_provider(provider)

# Creates a tracer from the global tracer provider
tracer = trace.get_tracer("my.tracer.name")

# Instrument BotoCore
BotocoreInstrumentor().instrument()

# Initialize S3 client
s3 = boto3.client("s3", region_name="us-east-1")

# Your bucket name
bucket_name = "my-example-bucket"

# Get bucket location (as an example of describing it)
try:
    response = s3.get_bucket_location(Bucket=bucket_name)
    region = response.get("LocationConstraint") or "us-east-1"
    print(f"Bucket '{bucket_name}' is in region: {region}")

    # Optionally, get bucket's creation date via list_buckets
    buckets = s3.list_buckets()
    for bucket in buckets["Buckets"]:
        if bucket["Name"] == bucket_name:
            print(f"Bucket created on: {bucket['CreationDate']}")
            break
except ClientError as e:
    print(f"Failed to describe bucket: {e}")
```

通过请求检测传出的 HTTP 调用

With X-Ray SDK

X-Ray Python SDK 通过修补请求库来通过请求跟踪传出的 HTTP 调用。有关更多信息，请参阅使用适用于 Python 的 X-Ray SDK 跟踪对下游 HTTP 网络服务的调用。在您的应用程序中，您可以使用`patch_all()`方法对所有库进行检测，也可以使用选择性地修补请求库。`patch(['requests'])`任何一个选项都会对`requests`库进行乐器，为通过`requests`的任何呼叫生成一个子分段。

With OpenTelemetry SDK

对于以下代码示例，您将需要以下依赖关系：

```
pip install opentelemetry-instrumentation-requests
```

以编程方式使用 Requests Instruction 来检测 OpenTelemetry 请求库，为其在应用程序中发出的 HTTP 请求生成跟踪。有关更多信息，请参阅OpenTelemetry 请求检测。以下示例演示了`requests`库工具。

```
from opentelemetry.instrumentation.requests import RequestsInstrumentor

# Instrument Requests
RequestsInstrumentor().instrument()

...

    example_session = requests.Session()
    example_session.get(url="https://example.com")
```

或者，您也可以对底层`urllib3`库进行检测以跟踪 HTTP 请求：

```
# pip install opentelemetry-instrumentation-urllib3
from opentelemetry.instrumentation.urllib3 import URLLib3Instrumentor

# Instrument urllib3
```

```
URLLib3Instrumentor().instrument()

...

    example_session = requests.Session()
    example_session.get(url="https://example.com")
```

对其他库的仪器支持

你可以在支持的库、框架、应用程序服务器和下找到支持 P OpenTelemetry ython 的库工具的完整列表。 JVMs

或者，您可以搜索 OpenTelemetry 注册表以了解是否 OpenTelemetry 支持检测。请查看注册表开始搜索。

手动创建跟踪数据

您可以在 Python 应用程序`xray_recorder`中使用创建区段和子区段。有关更多信息，请参阅手动检测 Python 代码。您也可以手动向跟踪数据添加注释和元数据。

使用 SDK 创建跨度 OpenTelemetry

使用 `start_as_current_span` API 启动跨度并将其设置为创建跨度。有关创建跨度的示例，请参阅创建跨度。跨度启动并处于当前作用域后，您可以通过添加属性、事件、异常、链接等向其添加更多信息。就像我们在 X-Ray 中使用分段和子分段一样，里面有不同类型的跨度。 OpenTelemetry只有`SERVER`种类跨度会转换为 X-Ray 分段，而其他跨度会转换为 X-Ray 子分段。

```
from opentelemetry import trace
from opentelemetry.trace import SpanKind

import time

tracer = trace.get_tracer("my.tracer.name")

# Create a new span to track some work
with tracer.start_as_current_span("parent", kind=SpanKind.SERVER) as parent_span:
    time.sleep(1)
```

```
    # Create a nested span to track nested work
    with tracer.start_as_current_span("child", kind=SpanKind.CLIENT) as child_span:
        time.sleep(2)
        # the nested span is closed when it's out of scope

    # Now the parent span is the current span again
    time.sleep(1)

    # This span is also closed when it goes out of scope
```

使用 OpenTelemetry SDK 向跟踪添加注释和元数据

X-Ray Python SDK 提供了单独的 APIs、put_annotationput_metadata用于向追踪添加注释和元数据的功能。在 OpenTelemetry SDK 中，注释和元数据只是通过 set_attribute API 添加的跨度上的属性。

您希望它们成为轨迹注释的 Span 属性会添加到保留键下，保留键的aws.xray.annotations值是注释的键值对列表。所有其他 span 属性都将成为转换后的分段或子分段的元数据。

此外，如果您使用的是 ADOT 收集器，则可以通过在收集器配置indexed_attributes中指定，来配置哪些跨度属性应转换为 X-Ray 注释。

以下示例演示了如何使用 OpenTelemetry SDK 向跟踪添加注释和元数据。

```
with tracer.start_as_current_span("parent", kind=SpanKind.SERVER) as parent_span:
    parent_span.set_attribute("TransactionId", "qwerty12345")
    parent_span.set_attribute("AccountId", "1234567890")

    # This will convert the TransactionId and AccountId to be searchable X-Ray
 annotations
    parent_span.set_attribute("aws.xray.annotations", ["TransactionId", "AccountId"])

    with tracer.start_as_current_span("child", kind=SpanKind.CLIENT) as child_span:

        # The MicroTransactionId will be converted to X-Ray metadata for the child
 subsegment
        child_span.set_attribute("MicroTransactionId", "micro12345")
```

Lambda 仪器

要在 X-Ray 上监控您的 lambda 函数，您可以启用 X-Ray 并向函数调用角色添加相应的权限。此外，如果您正在跟踪来自函数的下游请求，则需要使用 X-Ray Python SDK 来检测代码。

对 OpenTelemetry 于 X-Ray，建议在关闭 CloudWatch 应用程序信号的情况下使用应用信号 lambda 层。这将自动检测你的函数，并将为函数调用和来自你的函数的任何下游请求生成跨度。除了跟踪之外，如果您有兴趣使用应用程序信号来监控函数的运行状况，请参阅在 Lambda 上启用应用程序。

- 从 Lambda 层中找到您的函数所需的 Lambda 层 ARN 并将其AWS 添加。 OpenTelemetry ARNs
- 为您的函数设置以下环境变量。
  - `AWS_LAMBDA_EXEC_WRAPPER=/opt/otel-instrument`— 这将加载该函数的自动仪器
  - `OTEL_AWS_APPLICATION_SIGNALS_ENABLED=false`— 这将禁用应用程序信号监控

使用 Lambda 工具手动创建跨度

此外，你可以在函数中生成自定义跨度来跟踪工作。您可以仅将该`opentelemetry-api`软件包与应用程序信号 lambda 层自动检测结合使用。

1. 将`opentelemetry-api`作为依赖项包含在函数中
2. 以下代码片段是生成自定义跨度的示例

```
from opentelemetry import trace

# Get the tracer (auto-configured by the Application Signals layer)
tracer = trace.get_tracer(__name__)

def handler(event, context):
    # This span is a child of the layer's root span
    with tracer.start_as_current_span("my-custom-span") as span:
        span.set_attribute("key1", "value1")
        span.add_event("custom-event", {"detail": "something happened"})

        # Any logic you want to trace
        result = some_internal_logic()

    return {
        "statusCode": 200,
        "body": result
```

```
    }
```

迁移到 OpenTelemetry Ruby

要将 Ruby 应用程序从 X-Ray SDK 迁移到 OpenTelemetry 检测中，请使用以下代码示例和手动检测指南。

Sections

使用 SDK 手动分析您的解决方案

Tracing setup with X-Ray SDK

适用于 Ruby 的 X-Ruby SDK 要求您使用服务插件配置代码。

```
require 'aws-xray-sdk'

XRay.recorder.configure(plugins: [:ec2, :elastic_beanstalk])
```

Tracing setup with OpenTelemetry SDK

> **Note**
>
> 目前无法为 OpenTelemetry Ruby 配置 X-Ray 远程采样。

对于 Ruby on Rails 应用程序，请将配置代码放在 Rails 初始化器中。有关更多信息，请参阅 入门。对于所有手动检测的 Ruby 程序，必须使用该`OpenTelemetry::SDK.configure`方法来配置 OpenTelemetry Ruby SDK。

首先，安装以下软件包：

```
bundle add opentelemetry-sdk opentelemetry-exporter-otlp opentelemetry-propagator-
xray
```

接下来，通过程序初始化时运行的配置代码配置 OpenTelemetry SDK。建议您配置以下组件：

- `OTLP Exporter`— 将轨迹导出到 CloudWatch 代理和 OpenTelemetry 采集器时需要
- `An AWS X-Ray Propagator`— 需要将跟踪上下文传播到与 X-Ray 集成的 AWS 服务

```
require 'opentelemetry-sdk'
require 'opentelemetry-exporter-otlp'

# Import the gem containing the AWS X-Ray for OTel Ruby ID Generator and propagator
require 'opentelemetry-propagator-xray'

OpenTelemetry::SDK.configure do |c|
  c.service_name = 'my-service-name'

  c.add_span_processor(
    # Use the BatchSpanProcessor to send traces in groups instead of one at a time
    OpenTelemetry::SDK::Trace::Export::BatchSpanProcessor.new(
      # Use the default OLTP Exporter to send traces to the ADOT Collector
      OpenTelemetry::Exporter::OTLP::Exporter.new(
        # The OpenTelemetry Collector is running as a sidecar and listening on port
 4318
        endpoint:"http://127.0.0.1:4318/v1/traces"
      )
    )
  )

  # The X-Ray Propagator injects the X-Ray Tracing Header into downstream calls
  c.propagators = [OpenTelemetry::Propagator::XRay::TextMapPropagator.new]
end
```

OpenTelemetry SDKs 也有图书馆仪器的概念。启用这些功能将自动为 AWS SDK 等库创建跨度。OpenTelemetry 提供了启用所有库乐器或指定要启用哪些库乐器的选项。

要启用所有工具，请先安装软件包：opentelemetry-instrumentation-all

```
bundle add opentelemetry-instrumentation-all
```

接下来，更新配置以启用所有库工具，如下所示：

```
require 'opentelemetry/instrumentation/all'
...

OpenTelemetry::SDK.configure do |c|
   ...

  c.use_all() # Enable all instrumentations
end
```

OpenTelemetry SDKs 也有图书馆仪器的概念。启用这些功能将自动为 AWS SDK 等库创建跨度。OpenTelemetry 提供了启用所有库乐器或指定要启用哪些库乐器的选项。

要启用所有工具，请先安装软件包：opentelemetry-instrumentation-all

```
bundle add opentelemetry-instrumentation-all
```

接下来，更新配置以启用所有库工具，如下所示：

```
require 'opentelemetry/instrumentation/all'
...

OpenTelemetry::SDK.configure do |c|
   ...

  c.use_all() # Enable all instrumentations
end
```

跟踪传入的请求（Rails 工具）

With X-Ray SDK

使用 X-Rails SDK，可以在初始化时为 Rails 框架配置 X-Rails 跟踪。

示例 — config/initializers/aws _xray.rb

```
Rails.application.config.xray = {
  name: 'my app',
  patch: %I[net_http aws_sdk],
  active_record: true
}
```

With OpenTelemetry SDK

首先，安装以下软件包：

```
bundle add opentelemetry-instrumentation-rack opentelemetry-instrumentation-
rails opentelemetry-instrumentation-action_pack opentelemetry-instrumentation-
active_record opentelemetry-instrumentation-action_view
```

接下来，更新配置以为 Rails 应用程序启用检测功能，如下所示：

```
# During SDK configuration
OpenTelemetry::SDK.configure do |c|

  ...

  c.use 'OpenTelemetry::Instrumentation::Rails'
  c.use 'OpenTelemetry::Instrumentation::Rack'
  c.use 'OpenTelemetry::Instrumentation::ActionPack'
  c.use 'OpenTelemetry::Instrumentation::ActiveSupport'
  c.use 'OpenTelemetry::Instrumentation::ActionView'

  ...

end
```

AWS 软件开发工具包工具

With X-Ray SDK

为了检测来自 AWS SDK 的传出 AWS 请求，使用 X-Ray 修补了 AWS SDK 客户端，如下例所示：

```
require 'aws-xray-sdk'
require 'aws-sdk-s3'

# Patch AWS SDK clients
XRay.recorder.configure(plugins: [:aws_sdk])

# Use the instrumented client
s3 = Aws::S3::Client.new
s3.list_buckets
```

With OpenTelemetry SDK

AWS 适用于 Ruby V3 的 SDK 支持记录和发射 OpenTelemetry 轨迹。有关如何为服务客户端 OpenTelemetry 进行配置的信息，请参阅在 AWS SDK for Ruby 中配置可观察性功能。

检测传出 HTTP 调用

在对外部服务进行 HTTP 调用时，如果自动检测不可用或无法提供足够的细节，则可能需要手动检测调用。

With X-Ray SDK

为了检测下游调用，使用适用于 Ruby 的 X-Ray SDK 来修补您的应用程序使用的`net/http`库：

```
require 'aws-xray-sdk'

config = {
  name: 'my app',
  patch: %I[net_http]
}

XRay.recorder.configure(config)
```

With OpenTelemetry SDK

要使用启用net/http工具 OpenTelemetry，请先安装opentelemetry-instrumentation-net_http软件包：

```
bundle add opentelemetry-instrumentation-net_http
```

接下来，更新配置以启用net/http仪器，如下所示：

```
OpenTelemetry::SDK.configure do |c|
   ...

  c.use 'OpenTelemetry::Instrumentation::Net::HTTP'
  ...

end
```

对其他库的仪器支持

你可以在下面找到 OpenTelemetry Ruby 支持的库工具的完整列表。opentelemetry-ruby-contrib

或者，您可以搜索 OpenTelemetry 注册表以了解是否 OpenTelemetry 支持检测。有关更多信息，请参阅注册表。

手动创建跟踪数据

With X-Ray SDK

使用 X-Ray，该aws-xray-sdk软件包要求您手动创建区段及其子子分段来跟踪您的应用程序。您可能还为区段或子分段添加了 X-Ray 注释和元数据：

```
require 'aws-xray-sdk'
...

# Start a segment
segment = XRay.recorder.begin_segment('my-service')
```

```
# Add annotations (indexed key-value pairs)
segment.annotations[:user_id] = 'user-123'
segment.annotations[:payment_status] = 'completed'

# Add metadata (non-indexed data)
segment.metadata[:order] = {
  id: 'order-456',
  items: [
    { product_id: 'prod-1', quantity: 2 },
    { product_id: 'prod-2', quantity: 1 }
  ],
  total: 67.99
}

# Add metadata to a specific namespace
segment.metadata(namespace: 'payment') do |metadata|
  metadata[:transaction_id] = 'tx-789'
  metadata[:payment_method] = 'credit_card'
end

# Create a subsegment with annotations and metadata
segment.subsegment('payment-processing') do |subsegment1|
  subsegment1.annotations[:payment_id] = 'pay-123'
  subsegment1.metadata[:details] = { amount: 67.99, currency: 'USD' }

  # Create a nested subsegment
  subsegment1.subsegment('operation-2') do |subsegment2|
    # Do more work...
  end
end

# Close the segment
segment.close
```

With OpenTelemetry SDK

您可以使用自定义跨度来监控未被仪器库捕获的内部活动的性能。请注意，只有服务器类型的跨度才会转换为 X-Ray 分段，所有其他跨度都会转换为 X-Ray 子分段。默认情况下，跨度为。INTERNAL

首先，创建一个 Tracer 以生成跨度，你可以通过该OpenTelemetry.tracer_provider.tracer('<YOUR_TRACER_NAME>')方法获得跨度。

这将提供一个在您的应用程序 OpenTelemetry 配置中全局注册的 Tracer 实例。通常对整个应用程序使用一个 Tracer。创建一个 OpenTelemetry 示踪器并用它来创建跨度：

```
require 'opentelemetry-sdk'

...

# Get a tracer
tracer = OpenTelemetry.tracer_provider.tracer('my-application')

# Create a server span (equivalent to X-Ray segment)
tracer.in_span('my-application', kind: OpenTelemetry::Trace::SpanKind::SERVER) do |
span|
  # Do work...

  # Create nested spans of default kind INTERNAL will become an X-Ray subsegment
  tracer.in_span('operation-1') do |child_span1|
    # Set attributes (equivalent to X-Ray annotations and metadata)
    child_span1.set_attribute('key', 'value')

    # Do more work...
    tracer.in_span('operation-2') do |child_span2|
      # Do more work...
    end
  end
end
```

使用 OpenTelemetry SDK 向跟踪添加注释和元数据

使用`set_attribute`方法向每个跨度添加属性。请注意，默认情况下，所有这些跨度属性都将转换为 X-Ray 原始数据中的元数据。为确保将属性转换为注释而不是元数据，可以将该属性键添加到`aws.xray.annotations`属性列表中。有关更多信息，请参见启用自定义 X-Ray 注释。

```
# SERVER span will become an X-Ray segment
tracer.in_span('my-server-operation', kind: OpenTelemetry::Trace::SpanKind::SERVER)
 do |span|
    # Your server logic here
    span.set_attribute('attribute.key', 'attribute.value')
    span.set_attribute("metadataKey", "metadataValue")
    span.set_attribute("annotationKey1", "annotationValue")
```

```
    # Create X-Ray annotations
    span.set_attribute("aws.xray.annotations", ["annotationKey1"])
end
```

Lambda 手动检测

With X-Ray SDK

在 Lambda 上启用主动跟踪后，无需进行其他配置即可使用 X-Ray SDK。Lambda 将创建一个表示 Lambda 处理程序调用的区段，您无需任何额外配置即可使用 X-Ray SDK 创建子分段或仪器库。

With OpenTelemetry SDK

考虑以下 Lambda 函数代码示例（不带插入）：

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

要手动检测您的 Lambda，您需要：

1. 为你的 Lambda 添加以下宝石

   ```
   gem 'opentelemetry-sdk'
   gem 'opentelemetry-exporter-otlp'
   gem 'opentelemetry-propagator-xray'
   gem 'aws-distro-opentelemetry-exporter-xray-udp'
   gem 'opentelemetry-instrumentation-aws_lambda'
   gem 'opentelemetry-propagator-xray', '~> 0.24.0' # Requires version v0.24.0 or
    higher
   ```

2. 在 Lambda 处理程序之外初始化 OpenTelemetry 开发工具包。建议将 S OpenTelemetry DK 配置为：

   1. 一个带有 X-Ray UDP 跨度导出器的简单跨度处理器，用于向 Lambda 的 UDP X-Ray 端点发送跟踪

2. X-Ray Lambda 传播器
3. `service_name`要设置为 Lambda 函数名称的配置

3. 在您的 Lambda 处理程序类中，添加以下几行来检测您的 Lambda 处理程序：

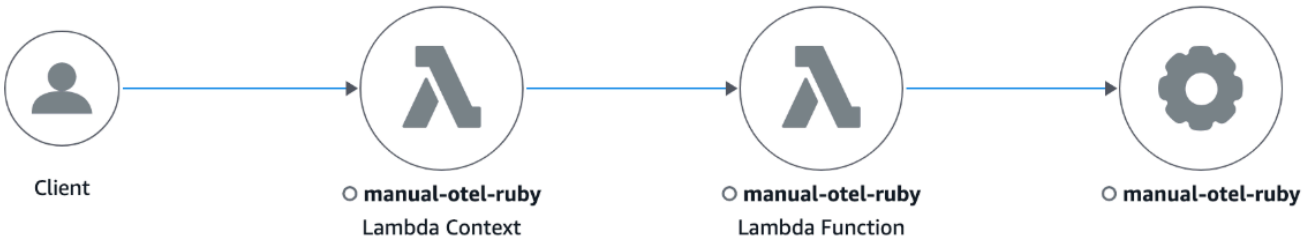
```
  class Handler
        extend OpenTelemetry::Instrumentation::AwsLambda::Wrap
        ...

        instrument_handler :process
    end
```

以下代码演示了进行必要更改后的 Lambda 函数。您可以创建其他自定义跨度来补充自动提供的跨度。

```
require 'json'
require 'opentelemetry-sdk'
require 'aws/distro/opentelemetry/exporter/xray/udp'
require 'opentelemetry/propagator/xray'
require 'opentelemetry/instrumentation/aws_lambda'

# Initialize OpenTelemetry SDK outside handler
OpenTelemetry::SDK.configure do |c|
  # Configure the AWS Distro for OpenTelemetry X-Ray Lambda exporter
  c.add_span_processor(
    OpenTelemetry::SDK::Trace::Export::SimpleSpanProcessor.new(
      AWS::Distro::OpenTelemetry::Exporter::XRay::UDP::AWSXRayUDPSpanExporter.new
    )
  )

  # Configure X-Ray Lambda propagator
  c.propagators = [OpenTelemetry::Propagator::XRay.lambda_text_map_propagator]

  # Set minimal resource information
  c.resource = OpenTelemetry::SDK::Resources::Resource.create({
    OpenTelemetry::SemanticConventions::Resource::SERVICE_NAME =>
 ENV['AWS_LAMBDA_FUNCTION_NAME']
  })
  c.use 'OpenTelemetry::Instrumentation::AwsLambda'
end

module LambdaFunctions
```

```
  class Handler
    extend OpenTelemetry::Instrumentation::AwsLambda::Wrap
    def self.process(event:, context:)
      "Hello!"
    end
    instrument_handler :process
  end
end
```

以下是用 Ruby 编写的带检测的 Lambda 函数的示例。



您也可以使用 Lambda 层为您的 Lambda OpenTelemetry 进行配置。有关更多信息，请参阅 OpenTelemetry AWS-Lambda 工具。

使用 AWS CloudFormation创建 X-Ray 资源

AWS X-Ray 与一项服务集成 AWS CloudFormation，该服务可帮助您对 AWS 资源进行建模和设置，从而减少创建和管理资源和基础架构所花费的时间。您可以创建一个描述所需所有 AWS 资源的模板，并为您预 AWS CloudFormation 置和配置这些资源。

使用时 AWS CloudFormation，您可以重复使用模板来一致且重复地设置 X-Ray 资源。只需描述一次您的资源，然后在多个 AWS 账户 区域中一遍又一遍地配置相同的资源。

X-Ray 和 AWS CloudFormation 模板

要为 X-Ray 和相关服务设置和配置资源，您必须了解AWS CloudFormation 模板。模板是 JSON 或 YAML 格式的文本文件。这些模板描述了您要在 AWS CloudFormation 堆栈中配置的资源。如果你不熟悉 JSON 或 YAML，可以使用 D AWS CloudFormation esigner 来帮助你开始使用 AWS CloudFormation 模板。有关更多信息，请参阅《AWS CloudFormation 用户指南》中的什么是 AWS CloudFormation Designer？。

X-Ray 支持在中创建`AWS::XRay::Group``AWS::XRay::SamplingRule`、和`AWS::XRay::ResourcePolicy`资源 AWS CloudFormation。有关更多信息（包括 JSON 和 YAML 模板的示例），请参阅《AWS CloudFormation 用户指南》中的 X-Ray 资源类型参考。

了解更多关于 AWS CloudFormation

要了解更多信息 AWS CloudFormation，请参阅以下资源：

- AWS CloudFormation
- AWS CloudFormation 用户指南
- AWS CloudFormation API 引用
- AWS CloudFormation 命令行界面用户指南

标记 X-Ray 采样规则和组

标签是可用于识别和组织 AWS 资源的单词或短语。您可以向每个资源添加多个标签。每个标签都包含您所定义的一个键和可选值。例如，标签键可能是 **`domain`**，标签值可能是 **`example.com`**。您可以根据添加的标签搜索和筛选您的资源。有关标签使用方法的更多信息，请参阅 AWS 一般参考中的标记 AWS 资源。

您可以使用标签对 CloudFront 分配强制执行基于标签的权限。有关更多信息，请参阅使用资源标签控制对 AWS 资源的访问。

> **Note**
>
> 标签编辑器和 AWS 资源组目前不支持 X-Ray 资源。您可以使用 AWS X-Ray 控制台或 API 添加和管理标签。

您可以使用 X-Ray 控制台、API、 AWS CLI SDKs、和对资源应用标签 AWS Tools for Windows PowerShell。有关更多信息，请参阅以下文档：

- X-Ray API — 请参阅 AWS X-Ray API 参考中介绍的以下操作：
  - ListTagsForResource
  - CreateSamplingRule
  - CreateGroup
  - TagResource
  - UntagResource
- AWS CLI — 参见《AWS CLI 命令参考》中的 xray
- SDKs — 参见 “文档” 页面上相应的 SDK AWS 文档

> **Note**
>
> 如果您无法在 X-Ray 资源上添加或更改标签，或者无法添加具有特定标签的资源，则可能没有权限执行此操作。要申请访问权限，请联系企业中拥有 X-Ray 管理员权限的 AWS 用户。

主题

标签限制

以下限制适用于标签。

- 每个资源的标签数上限 – 50
- 最大密钥长度 – 128 个 Unicode 字符
- 最大值长度 – 256 个 Unicode 字符
- 键和值的有效值 – a-z、A-Z、0-9、空格和以下字符：_ . : / = + - 和 @
- 标签键和值区分大小写。
- 请不要使用 `aws:` 作为键的前缀；它保留为供 AWS 使用。

> **Note**
>
> 无法编辑或删除系统标签。

在控制台中管理标签

您可以在创建 X-Ray 组或采样规则时添加可选标签。稍后也可以在控制台中更改或删除标签。

以下过程介绍了如何在 X-Ray 控制台中为集群和采样规则添加、编辑和删除标签。

主题

向新组添加标签（控制台）

创建新的 X-Ray 组时，可以在创建组页面上添加可选标签。

1. 登录 AWS Management Console 并在https://console.aws.amazon.com/xray/家中打开 X-Ray 控制台。
2. 在导航窗格中，展开配置，然后选择组。
3. 选择创建群组。
4. 在创建组页面上，为组指定名称和筛选表达式。有关这些属性的更多信息，请参阅配置组。
5. 在标签中，输入一个标签键和可选的标签值。例如，可以输入标签键“**Stage**”和标签值“**Production**”以指示该组用于生产。添加标签时会出现一个新行，供您根据需要添加另一个标签。请参阅本主题中的 标签限制，了解标签的限制。
6. 在添加完标签后，请选择创建组。

向新采样规则添加标签（控制台）

创建新的 X-Ray 采样规则时，可以在创建采样规则页面上添加标签。

1. 登录 AWS Management Console 并在https://console.aws.amazon.com/xray/家中打开 X-Ray 控制台。
2. 在导航窗格中，展开配置，然后选择采样。
3. 选择创建采样规则。
4. 在创建采样规则页面上，指定名称、优先级、限制、匹配标准和匹配属性。有关这些属性的更多信息，请参阅配置采样规则。
5. 在标签中，输入一个标签键和可选的标签值。例如，可以输入标签键“**Stage**”和标签值“**Production**”以指示采购规则用于生产。添加标签时会出现一个新行，供您根据需要添加另一个标签。请参阅本主题中的 标签限制，了解标签的限制。
6. 在添加完标签后，请选择创建采样规则。

编辑或删除某个组的标签（控制台）

您可以在编辑组页面上更改或删除应用于某个 X-Ray 组的标签。

1. 登录 AWS Management Console 并在https://console.aws.amazon.com/xray/家中打开 X-Ray 控制台。

2. 在导航窗格中，展开配置，然后选择组。
3. 在组表中，选择相应组的名称。
4. 在编辑组页面的标签中，编辑标签键和值。标签键不得重复。标签值是选填，如果需要可以删除。有关编辑组页面上其他属性的更多信息，请参阅 配置组。请参阅本主题中的 标签限制，了解标签的限制。
5. 若要删除标签，请选择标签右侧的 X。
6. 编辑或删除完标签后，请选择更新组。

编辑或删除采样规则标签（控制台）

您可以在编辑采样规则页面上更改或删除应用于 X-Ray 采样规则的标签。

1. 登录 AWS Management Console 并在https://console.aws.amazon.com/xray/家中打开 X-Ray 控制台。
2. 在导航窗格中，展开配置，然后选择采样。
3. 在采样规则表中，选择采样规则的名称。
4. 在标签中，编辑标签键和值。标签键不得重复。标签值是选填，如果需要可以删除。有关编辑采样规则页面上其他属性的更多信息，请参阅 配置采样规则。请参阅本主题中的 标签限制，了解标签的限制。
5. 若要删除标签，请选择标签右侧的 X。
6. 编辑或删除完标签后，选择更新采样规则。

管理中的标签 AWS CLI

您可以在创建 X-Ray 组或采样规则时添加标签。您也可以使用 AWS CLI 来创建和管理标签。要更新现有群组或采样规则上的标签，请使用 AWS X-Ray 控制台TagResource或或UntagResource APIs。

主题

- 向新的 X-Ray 组或采样规则添加标签 (CLI)
- 向现有资源添加标签 (CLI)
- 列出资源上的标签 (CLI)
- 从资源中删除标签 (CLI)

向新的 X-Ray 组或采样规则添加标签 (CLI)

请使用以下命令之一，在创建新的 X-Ray 组或采样规则时添加可选标签。

- 要向新组添加标签，请运行以下命令，*group_name*替换为您的组名称、*mydomain.com*服务的终端节点、*key_name*标签密钥以及可选*value*的标签值。有关如何创建组的更多信息，请参阅组。

```
aws xray create-group \
   --group-name "group_name" \
   --filter-expression "service(\"mydomain.com\") {fault OR error}" \
   --tags [{"Key": "key_name","Value": "value"},{"Key": "key_name","Value": "value"}]
```

  示例如下：

```
aws xray create-group \
   --group-name "AdminGroup" \
   --filter-expression "service(\"mydomain.com\") {fault OR error}" \
   --tags [{"Key": "Stage","Value": "Prod"},{"Key": "Department","Value": "QA"}]
```

- 要向新的采样规则添加标签，请运行以下命令，*key_name*替换为标签键，也可以*value*替换为标签值。此命令将 `--sampling-rule` 参数中的值指定为 JSON 文件。有关如何创建采样规则的更多信息，请参阅采样规则。

```
aws xray create-sampling-rule \
   --cli-input-json file://file_name.json
```

  以下是`--cli-input-json`参数指定的 JSON 文件的*file_name.json*内容。

```
{
    "SamplingRule": {
        "RuleName": "rule_name",
        "RuleARN": "string",
        "ResourceARN": "string",
        "Priority": integer,
        "FixedRate": double,
        "ReservoirSize": integer,
        "ServiceName": "string",
        "ServiceType": "string",
        "Host": "string",
        "HTTPMethod": "string",
        "URLPath": "string",
```

```
        "Version": integer,
        "Attributes": {"attribute_name": "value","attribute_name": "value"...}
    }
    "Tags": [
           {
               "Key":"key_name",
               "Value":"value"
           },
           {
               "Key":"key_name",
               "Value":"value"
           }
          ]
}
```

以下命令是一个示例。

```
aws xray create-sampling-rule \
   --cli-input-json file://9000-base-scorekeep.json
```

以上是由 `--cli-input-json` 参数指定的示例 `9000-base-scorekeep.json` 文件的内容。

```
{
    "SamplingRule": {
        "RuleName": "base-scorekeep",
        "ResourceARN": "*",
        "Priority": 9000,
        "FixedRate": 0.1,
        "ReservoirSize": 5,
        "ServiceName": "Scorekeep",
        "ServiceType": "*",
        "Host": "*",
        "HTTPMethod": "*",
        "URLPath": "*",
        "Version": 1
    }
    "Tags": [
           {
               "Key":"Stage",
               "Value":"Prod"
           },
           {
```

```
                "Key":"Department",
                "Value":"QA"
            }
           ]
}
```

向现有资源添加标签 (CLI)

可以运行 `tag-resource` 命令向现有 X-Ray 组或采样规则添加标签。此方法可能比通过运行 `update-group` 或 `update-sampling-rule` 来添加标签更简单。

若要向组或采样规则添加标签，请运行以下命令，将 ARN 替换为资源的 ARN，并指定想要添加的标签的键和可选值。

```
aws xray tag-resource \
   --resource-arn "ARN" \
   --tag-keys [{"Key":"key_name","Value":"value"}, {"Key":"key_name","Value":"value"}]
```

示例如下：

```
aws xray tag-resource \
   --resource-arn "arn:aws:xray:us-east-2:01234567890:group/AdminGroup" \
   --tag-keys [{"Key": "Stage","Value": "Prod"},{"Key": "Department","Value": "QA"}]
```

列出资源上的标签 (CLI)

可以运行 `list-tags-for-resource` 命令列出 X-Ray 组或采样规则上的标签。

若要列出与某个组或采样规则关联的标签，请运行以下命令，将 ARN 替换为资源的 ARN。

```
aws xray list-tags-for-resource \
   --resource-arn "ARN"
```

示例如下：

```
aws xray list-tags-for-resource \
   --resource-arn "arn:aws:xray:us-east-2:01234567890:group/AdminGroup"
```

从资源中删除标签 (CLI)

可以运行 `untag-resource` 命令删除 X-Ray 组或采样规则上的标签。

若要删除组或采样规则上的标签，请运行以下命令，将 ARN 替换为资源的 ARN，并指定想要添加的标签的键。

可以使用 `untag-resource` 命令仅删除整个标签。若要删除标签值，请使用 X-Ray 控制台，或删除标签再添加具有相同键但值不同或为空的新标签。

```
aws xray untag-resource \
   --resource-arn "ARN" \
   --tag-keys ["key_name","key_name"]
```

示例如下：

```
aws xray untag-resource \
   --resource-arn "arn:aws:xray:us-east-2:01234567890:group/group_name" \
   --tag-keys ["Stage","Department"]
```

基于标签控制对 X-Ray 资源的访问

您可以将标签附加到 X-Ray 组或采样规则，或在发给 X-Ray 请求中传递标签。要基于标签控制访问，您需要使用 `xray: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。要了解有关这些条件键的更多信息，请参阅使用 AWS 资源标签控制对资源的访问权限。

要查看基于身份的策略（用于根据资源上的标签来限制对该资源的访问）的示例，请参阅根据标签管理对 X-Ray 组和采样规则的访问权限。

故障排除 AWS X-Ray

本主题列出了您在使用 X-Ray API、控制台或时可能遇到的常见错误和问题 SDKs。如果您发现某个问题未在此处列出，可以使用此页上的反馈按钮来报告。

Sections

X-Ray 跟踪地图和跟踪详情页面

如果您在使用 X-Ray 跟踪地图和跟踪详情页面时遇到问题，以下各节可以提供帮助：

我没有看到我所有的日 CloudWatch 志

如何配置日志，使其显示在 X-Ray 跟踪地图和跟踪详情页面中，具体取决于服务。

- 如果已在 API Gateway 中启用了日志记录，则会显示 API Gateway 日志。

并非所有服务地图节点都支持查看关联的日志。查看以下节点类型的日志：

- Lambda 上下文
- Lambda 函数
- API Gateway 阶段
- Amazon ECS 集群
- Amazon ECS 实例
- Amazon ECS 服务
- Amazon ECS 任务
- Amazon EKS 集群
- Amazon EKS 命名空间
- Amazon EKS 节点

- Amazon EKS 容器组（pod）
- Amazon EKS 服务

我未在 X-Ray 跟踪地图上看到我的所有警报

如果与节点关联的任何警报都处于 ALARM 状态，则 X-Ray 跟踪地图仅显示该节点的提示图标。

跟踪地图使用以下逻辑将警报与节点关联：

- 如果该节点代表一项 AWS 服务，则所有与该服务关联的命名空间的警报都将与该节点相关联。例如，类型`AWS::Kinesis`为的节点与所有基于 CloudWatch 命名空间中指标的警报相关联`AWS/Kinesis`。
- 如果该节点代表一种 AWS 资源，则会关联该特定资源的警报。例如，名为 “MyTable” `AWS::DynamoDB::Table` 的节点会链接到基于命名空间`AWS/DynamoDB`且`TableName`维度设置为的指标的所有警报`MyTable`。
- 如果节点的类型未知（由名称周围的虚线边框标识），则任何警报均不会与该节点关联。

我没有在跟踪地图上看到某些 AWS 资源

并非每个 AWS 资源都由专用节点表示。有些 AWS 服务由一个节点表示，用于向该服务发出的所有请求。将显示以下资源类型，并且每个资源对应一个节点：

- `AWS::DynamoDB::Table`
- `AWS::Lambda::Function`

  Lambda 函数由两个节点表示：一个表示 Lambda 容器，另一个表示函数。这有助于识别 Lambda 函数的冷启动问题。Lambda 容器节点与警报和控制面板的关联方式与 Lambda 函数节点与警报和控制面板的关联方式相同。
- `AWS::ApiGateway::Stage`
- `AWS::SQS::Queue`
- `AWS::SNS::Topic`

跟踪地图包含太多节点

使用 X-Ray 组将您的地图分成多个地图。有关更多信息，请参阅对组使用筛选表达式。

适用 Java 的 X-Ray 开发工具包

错误：话题 “Thread-1” 出现异常 com.amazonaws.xray.exceptions SegmentNotFoundException: 无法开始名为 “AmazonSNS” 的子区段：找不到区段。

此错误表示 X-Ray SDK 尝试录制拨出的呼叫 AWS，但找不到打开的分段。这可在以下情况下发生：

- servlet 筛选条件未配置 - X-Ray SDK 会使用名为 `AWSXRayServletFilter` 的筛选条件为传入请求创建分段。配置 servlet 筛选条件来检测传入请求。
- 您正在 servlet 代码外部使用检测过的客户端 - 如果您在启动代码或并非运行用于响应传入请求的其他代码中，使用检测过的客户端来发出调用，则必须手动创建一个分段。有关示例，请参阅 检测启动代码。
- 您正在工作线程中使用检测过的客户端 - 当您创建新线程时，X-Ray 记录器会丢失对打开的分段的引用。您可以使用 `getTraceEntity` 和 `setTraceEntity` 方法来获取对当前分段或子分段的引用 (`Entity`)，并将其传递回线程内部的记录器。有关示例，请参阅 在工作线程中使用检测的客户端。

适用于 Node.js 的 X-Ray 软件开发工具包

问题：CLS 无法与 Sequelize 一起使用

使用 `cls` 方法，将 X-Ray SDK for Node.js 命名空间传递到 Sequelize。

```
var AWSXRay = require('aws-xray-sdk');
const Sequelize = require('sequelize');
Sequelize.cls = AWSXRay.getNamespace();
const sequelize = new Sequelize(...);
```

问题：CLS 无法与 Bluebird 一起使用

使用 `cls-bluebird` 实现 Bluebird 与 CLS 配合工作。

```
var AWSXRay = require('aws-xray-sdk');
var Promise = require('bluebird');
var clsBluebird = require('cls-bluebird');
clsBluebird(AWSXRay.getNamespace());
```

X-Ray 进程守护程序

问题：进程守护程序使用错误的凭证

守护程序使用 AWS SDK 加载凭证。如果您使用多种方法提供凭证，将使用优先顺序最高的方法。参阅 运行进程守护程序 了解更多信息。

的文档历史记录 AWS X-Ray

下表描述了对的文档所做的重要更改 AWS X-Ray。要获得本文档的更新通知，您可以订阅 RSS 源。

文档最新更新时间：2024 年 3 月 07 日

变更	说明	日期
新增功能	从 X-Ray 迁移到 OpenTelemetry. 有关更多信息，请参阅从 X-Ray 仪器迁移到 OpenTelemetry 仪器。	2025 年 6 月 13 日
新增功能	AWS X-Ray 现在支持交易搜索。有关更多信息，请参阅 Transaction Search。	2024 年 11 月 21 日
新增功能	AWS X-Ray 现在支持 OpenTelemetry 协议 (OTLP) 端点。有关更多信息，请参阅 OpenTelemetry。	2024 年 11 月 21 日
新增功能	X-Ray 现在可以记录数据事件`PutTraceSegments` ，包括`GetTraceSummaries` 、和`BatchGetTraces` AWS CloudTrail。X-Ray 现在还会将`GetSamplingStatisticSummaries` 管理事件记录到 CloudTrail。有关更多信息，请参阅使用记录 X-Ray API 调用 AWS CloudTrail。	2024 年 3 月 7 日
新增功能	X-Ray 现在支持通过 OpenTelemetry 或任何其他符合 W3C 跟踪上下文规范的框架 IDs 创建的跟踪。有关更多	2023 年 10 月 25 日

	信息，请参阅向 X-Ray 发送跟踪数据。	
新增功能	现在，您可以配置 Amazon SNS 主动跟踪，从而跟踪和分析请求在通过您的 Amazon SNS 主题传输时的情形。有关更多信息，请参阅 Amazon SNS 和。 AWS X-Ray	2023 年 2 月 8 日
更新了 X-Ray SDK for Node.js 主题	添加了有关使用 适用于 JavaScript 的 AWS SDK V3 对客户端进行检测的详细信息。有关详细信息，请参阅使用适用于 Node.js 的 X-Ray SD AWS K 跟踪 SDK 调用。	2023 年 2 月 7 日
更新了 IAM 托管式策略详细信息	为 `AWSXRayReadOnlyAccess` 、`AWSXRayFullAccess` 和 `AWSXrayCrossAccountSharingConfiguration` 托管式策略添加了 IAM 权限以实现跨账户可观测性。有关详细信息，请参阅适用于 X-Ray 的 IAM 托管式策略。	2023 年 2 月 7 日
新增功能	AWS X-Ray 现在支持跨账户可观察性，使您能够监控跨多个账户的应用程序并对其进行故障排除。 AWS 区域有关详细信息，请参阅跨账户跟踪。	2022 年 11 月 27 日

新增功能	现在，您可以查看消息创建者、Amazon SQS 队列和使用器之间关联的跟踪，提供事件驱动型应用程序发送的跟踪的互联视图。有关更多信息，请参阅跟踪事件驱动型应用程序。	2022 年 11 月 20 日
更新了 IAM 托管式策略详细信息	向 `AWSXRayReadOnlyAccess` 托管式策略添加了列出资源策略的 IAM 权限。有关详细信息，请参阅适用于 X-Ray 的 IAM 托管式策略。	2022 年 11 月 15 日
更新了 IAM 控制台权限和托管策略详细信息	更新了 X-Ray 控制台使用的 IAM 权限集，以及 `AWSXRayReadOnlyAccess` 托管式策略的说明。有关详细信息，请参阅使用 X-Ray 控制台。	2022 年 11 月 11 日
为 Rub AWS y 添加了发行 OpenTelemetry 版	AWS Distro for OpenTelemetry (ADOT) 提供了一组单一的开源 APIs、库和代理，用于收集分布式跟踪和指标。ADOT Ruby 使您能够为 X-Ray 和其他跟踪后端检测 Ruby 应用程序。有关更多信息，请参阅 OpenTelemetry Ruby AWS 发行版。	2022 年 2 月 7 日
新增功能	现在，您可以从 CloudWatch 控制台查看跟踪和配置 X-Ray。有关更多信息，请参阅 X-Ray 控制台。	2022 年 1 月 24 日

集成 CloudWatch RUM	借助 AWS X-Ray 和 CloudWatch RUM，您可以分析和调试从应用程序的最终用户到下游 AWS 托管服务的请求路径。有关更多信息，请参阅 CloudWatch RUM 和 AWS X-Ray。	2021 年 12 月 3 日
的集成 AWS 发行版 OpenTelemetry	AWS Distro for OpenTelemetry (ADOT) 提供了一组单一的开源 APIs、库和代理，用于收集分布式跟踪和指标。ADOT 使您能够为 X-Ray 和其他跟踪后端检测应用程序。有关更多信息，请参阅检测应用程序。	2021 年 9 月 23 日
新增功能	AWS X-Ray 现已与亚马逊 Virtual Private Cloud 集成，使您的亚马逊 VPC 中的资源无需通过公共互联网即可与 X-Ray 服务进行通信。有关更多信息，请参阅AWS X-Ray 与 VPC 终端节点一起使用。	2021 年 5 月 20 日
新增功能	AWS X-Ray 现在与集成 AWS CloudFormation，使您能够配置和配置 X-Ray 资源。有关更多信息，请参阅使用创建 X-Ray 资源 CloudFormation。	2021 年 5 月 6 日
新增功能	AWS X-Ray 现在与 Amazon EventBridge 集成，可以追踪通过的事件 EventBridge。这为用户提供了更全面的系统视图。有关更多信息，请参阅 Amazon EventBridge 和 AWS X-Ray。	2021 年 3 月 2 日

ECR 添加了进程守护程序	现在可以从 Amazon ECR 下载进程守护程序。有关更多信息，请参阅下载进程守护程序。	2021 年 3 月 1 日
新增功能	AWS X-Ray 现在支持向 Amazon 发送与见解相关的通知 EventBridge。这允许您使用自动对见解采取操作 EventBridge。有关更多信息，请参阅见解通知。	2020 年 10 月 15 日
添加了可下载的进程守护程序	AWS X-Ray 引入了 Linux ARM64 的支持守护程序。有关更多信息，请参阅 AWS X-Ray daemonbrazil ws	2020 年 10 月 1 日
新增功能	AWS X-Ray 现在支持与 Amazon Synthetics 的主动集 CloudWatch 成。让您可以查看与 Synthetics Canary 客户端节点有关的详细信息，例如响应时间和状态。您还可以根据来自 Synthetics Canary 客户端节点的信息在 Analytics 控制台中进行分析。有关更多信息，请参阅使用 X-Ray 调试 CloudWatch 合成金丝雀。	2020 年 9 月 24 日
新增功能	AWS X-Ray 现在支持对的跟踪 end-to-end工作流程 AWS Step Functions。您可以可视化状态机的组件、确定性能瓶颈以及对导致错误的请求进行故障排除。有关更多信息，请参阅AWS Step Functions 和 AWS X-Ray。	2020 年 9 月 14 日

新增功能	AWS X-Ray 引入洞察力，持续分析您账户中的跟踪数据，以识别应用程序中出现的紧急问题。见解会记录事件并跟踪事件影响，直到问题得到解决。有关更多信息，请参阅在AWS X-Ray 控制台中使用见解	2020 年 9 月 3 日
新增功能	AWS X-Ray 引入了 Java 自动检测代理，使客户无需修改现有的基于 Java 的应用程序即可收集跟踪数据。现在，您可以跟踪基于 Java Web 和 servlet 的应用程序，只需进行最少的配置更改且无需更改代码。有关更多信息，请参阅适用于 Java 的AWS X-Ray 自动检测代理。	2020 年 9 月 3 日
新增功能	AWS X-Ray 已在 X-Ray 控制台中添加了一个新的 “群组” 页面，以帮助简化追踪组的创建和管理。有关更多信息，请参阅在 X-Ray 控制台中配置组。	2020 年 8 月 24 日
新增功能	AWS X-Ray 现在允许您向群组和采样规则添加标签。您还可以基于标签来控制对组和采样规则的访问。有关更多信息，请参阅标记 X-Ray 采样规则和组和基于标签管理对 X-Ray 组和采样规则的访问。	2020 年 8 月 24 日

本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。