開發人員指南

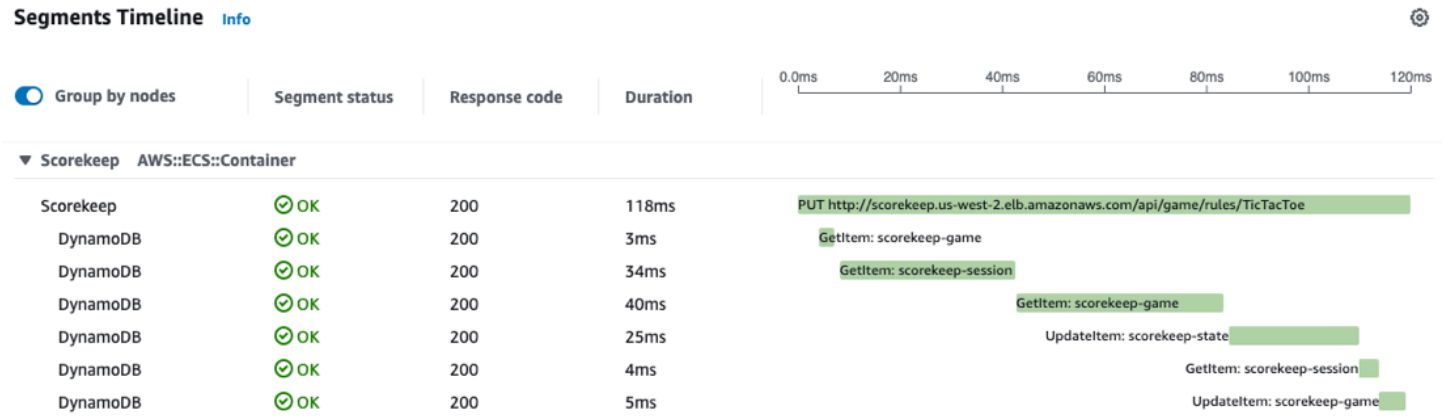
AWS X-Ray

AWS X-Ray: 開發人員指南

Table of Contents

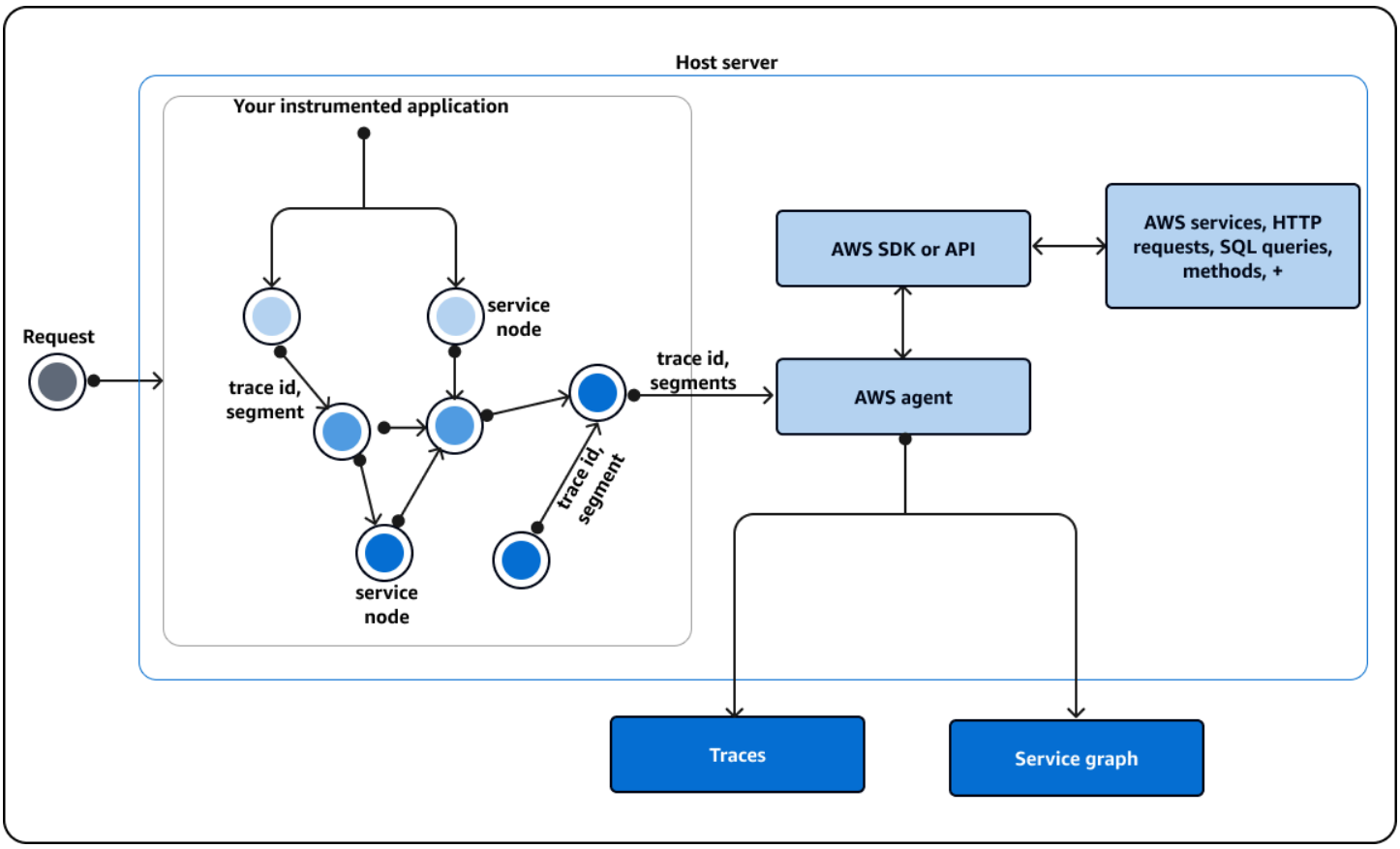
什麼是 AWS X-Ray？

AWS X-Ray 是一種 服務，可收集應用程式所提供服務請求的資料，並提供可用來檢視、篩選和深入了解該資料的工具，以識別問題和最佳化的機會。對於對應用程式的任何追蹤請求，您不僅可以查看請求和回應的詳細資訊，還可以查看應用程式對下游 AWS 資源、微服務、資料庫和 Web APIs發出的呼叫的詳細資訊。



AWS X-Ray 除了已與 X-Ray 整合 AWS 服務 的應用程式使用之外， 也會接收來自應用程式的追蹤。檢測您的應用程式涉及傳送傳入和傳出請求的追蹤資料，以及應用程式中的其他事件，以及每個請求的中繼資料。許多檢測案例僅需要組態變更。例如，您可以檢測所有傳入的 HTTP 請求和 AWS 服務對 Java 應用程式進行的下游呼叫。有數個 SDKs、代理程式和工具可用於檢測您的應用程式以進行 X-Ray 追蹤。如需詳細資訊，請參閱檢測您的應用程式。

AWS 服務 與 X-Ray 整合的 可以將追蹤標頭新增至傳入請求、將追蹤資料傳送至 X-Ray，或執行 X-Ray 協助程式。例如， AWS Lambda 可以將有關請求的追蹤資料傳送到 Lambda 函數，並在工作者上執行 X-Ray 協助程式，以更輕鬆地使用 X-Ray SDK。



每個用戶端 SDK 不會將追蹤資料直接傳送到 X-Ray，而是將 JSON 區段文件傳送到監聽 UDP 流量的協助程式程序。X-Ray 協助程式會緩衝佇列中的區段，並分批上傳至 X-Ray。協助程式適用於 Linux、Windows 和 macOS，並包含在 AWS Elastic Beanstalk 和 AWS Lambda 平台上。

X-Ray 使用來自支援雲端應用程式 AWS 的資源的追蹤資料來產生詳細的追蹤映射。追蹤映射會顯示用戶端、您的前端服務和後端服務，您的前端服務會呼叫這些服務來處理請求並保留資料。使用追蹤映射來識別瓶頸、延遲尖峰和其他要解決的問題，以改善應用程式的效能。


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

X-Ray 入門

若要使用 X-Ray，請執行下列步驟：

1. 檢測您的應用程式，這可讓 X-Ray 追蹤您的應用程式如何處理請求。

   - 使用 X-Ray SDKs、X-Ray APIs ADOT或 CloudWatch Application Signals 將追蹤資料傳送至 X-Ray。如需使用哪個界面的詳細資訊，請參閱 選擇界面。

   如需檢測的詳細資訊，請參閱 檢測您的應用程式 AWS X-Ray。

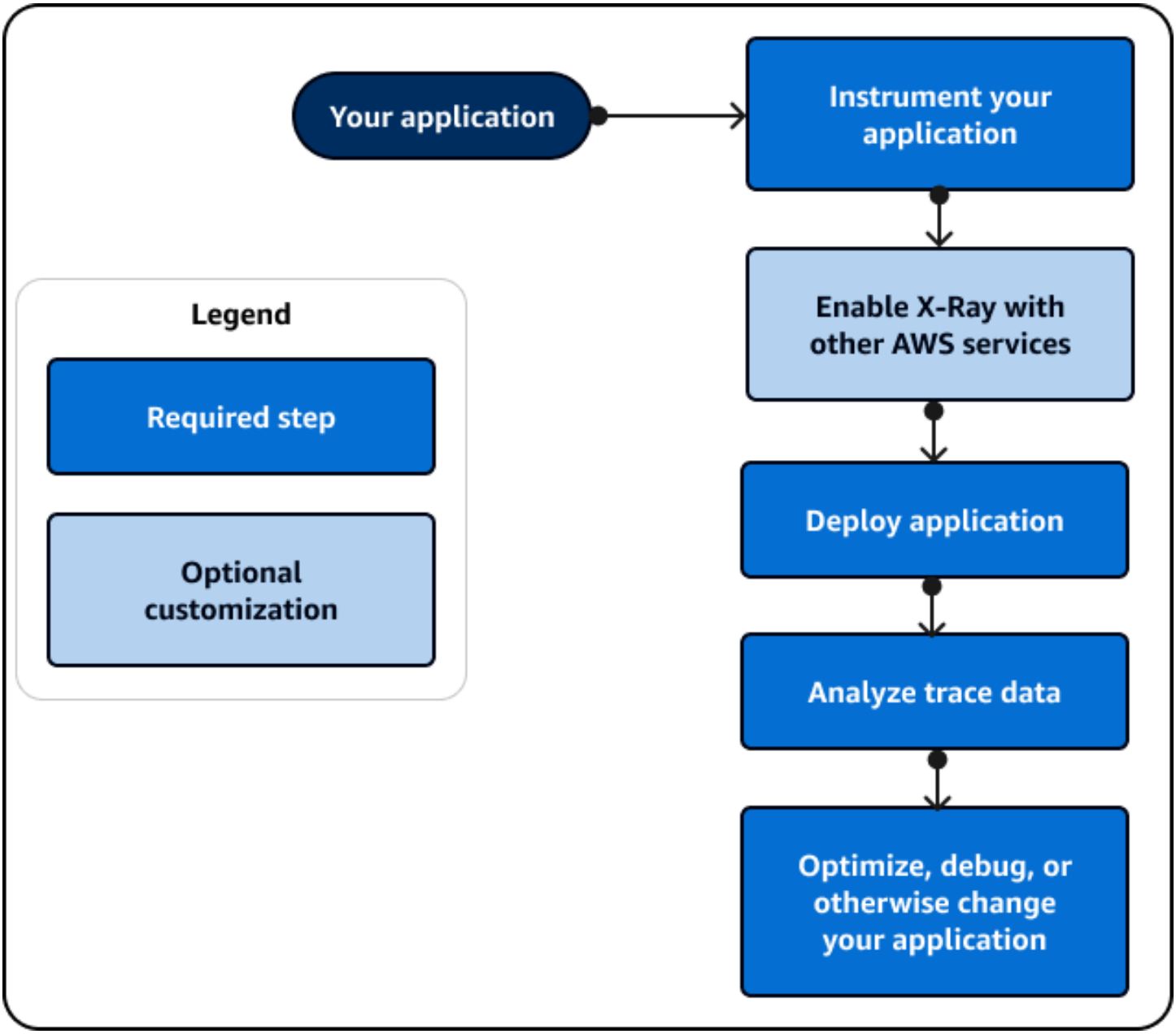
2. （選用） 將 X-Ray 設定為與其他 搭配 AWS 服務 ，以整合 X-Ray。您可以取樣追蹤並將標頭新增至傳入請求、執行代理程式或收集器，以及自動將追蹤資料傳送至 X-Ray。如需詳細資訊，請參閱AWS X-Ray 與其他 整合 AWS 服務。

3. 部署經檢測的應用程式。當您的應用程式收到請求時，X-Ray 開發套件會記錄追蹤、區段和子區段資料。在此步驟中，您可能還必須設定 IAM 政策並部署代理程式或收集器。

   - 如需在不同平台上使用 AWS Distro for OpenTelemetry (ADOT) SDK 和 CloudWatch 代理程式部署應用程式的範例指令碼，請參閱 Application Signals 示範指令碼。
   - 如需使用 X-Ray SDK 和 X-Ray 協助程式部署應用程式的範例指令碼，請參閱 AWS X-Ray 範例應用程式。

4. （選用） 開啟主控台以檢視和分析資料。您可以查看追蹤映射、服務映射等的 GUI 表示法，以檢查應用程式的運作方式。使用 主控台中的圖形資訊來最佳化、偵錯和了解您的應用程式。如需選擇主控台的詳細資訊，請參閱 使用主控台。

下圖顯示如何開始使用 X-Ray：



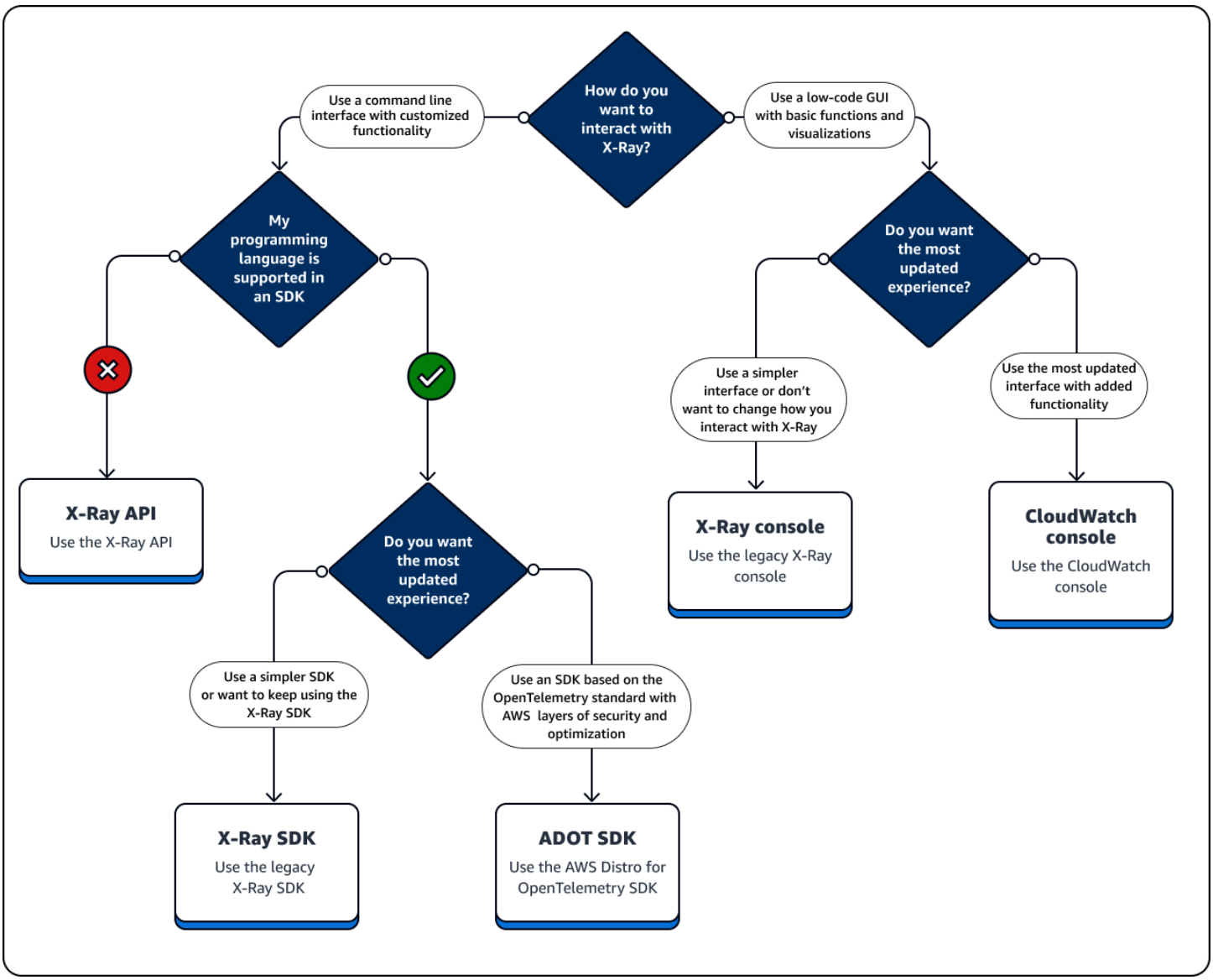
如需 主控台中可用資料和映射的範例，請啟動已進行檢測以產生追蹤資料的範例應用程式。在幾分鐘內，您可以產生流量、將區段傳送至 X-Ray，以及檢視追蹤和服務地圖。

選擇界面

AWS X-Ray 可以提供洞見，了解您的應用程式的運作方式，以及與其他 服務和資源的互動方式。在您檢測或設定應用程式之後，X-Ray 會在應用程式提供請求時收集追蹤資料。您可以分析此追蹤資料，以識別效能問題、疑難排解錯誤，以及最佳化您的 資源。本指南說明如何使用下列準則與 X-Ray 互動：

- AWS Management Console 如果您想要快速入門，請使用 ，或者可以使用預先建置的視覺化來執行基本任務。
  - 選擇 Amazon CloudWatch 主控台以取得包含所有 X-Ray 主控台功能的最新使用者體驗。
  - 如果您想要更簡單的界面或不想變更與 X-Ray 互動的方式，請使用 X-Ray 主控台。
- 如果您需要比 更多的自訂追蹤、監控或記錄功能，請使用 開發套件 AWS Management Console 。
  - ADOT 如果您想要以開放原始碼 SDK 為基礎，並新增 AWS 安全與最佳化層級的廠商無關 OpenTelemetry SDK，請選擇 SDK。
  - 如果您想要更簡單的 SDK 或不想要更新您的應用程式碼，請選擇 X-Ray 開發套件。
- 如果開發套件不支援應用程式的程式設計語言，請使用 X-Ray API 操作。

下圖可協助您選擇如何與 X-Ray 互動：



探索界面類型

- 使用 SDK
- 使用主控台
- 使用 X-Ray API

使用 SDK

如果您想要使用命令列界面，或需要比 中可用的更多自訂追蹤、監控或記錄功能，請使用 開發套件 AWS Management Console。您也可以使用 AWS SDK 來開發使用 X-Ray APIs程式。您可以使用 AWS Distro for OpenTelemetry (ADOT) SDK 或 X-Ray SDK。

如果您使用 SDK，您可以在檢測應用程式和設定收集器或代理程式時，將自訂新增至工作流程。您可以使用 SDK 來執行以下無法使用 執行的任務 AWS Management Console：

- 發佈自訂指標 – 高解析度低至 1 秒的指標範例，使用多個維度來新增指標的相關資訊，並將資料點彙總至統計資料集。
- 自訂您的收集器 – 自訂收集器任何部分的組態，包括接收者、處理器、匯出器和連接器。
- 自訂檢測 – 自訂區段和子區段、新增自訂鍵值對做為屬性，以及建立自訂指標。
- 以程式設計方式建立和更新抽樣規則。

如果您想要彈性地使用標準 OpenTelemetry SDK 並新增 AWS 安全與最佳化層級，請使用 ADOT SDK。 AWS Distro for OpenTelemetry (ADOT) SDK 是與廠商無關的套件，允許與其他廠商和非AWS服務後端整合，而不必重新檢測程式碼。

如果您已使用 X-Ray 開發套件、僅與 AWS 後端整合，而且不想變更您與 X-Ray 或應用程式程式碼互動的方式，請使用 X-Ray 開發套件。

如需每個功能的詳細資訊，請參閱 選擇 AWS Distro for OpenTelemetry 和 X-Ray SDKs。

使用 ADOT SDK

SDK ADOT是一組開放原始碼 APIs、程式庫和代理程式，可將資料傳送至後端服務。 ADOT 支援 AWS，整合多個後端和代理程式，並提供OpenTelemetry社群維護的大量開放原始碼程式庫。使用 ADOT 開發套件來檢測您的應用程式，並收集日誌、中繼資料、指標和追蹤。您也可以使用 ADOT來監控服務，並根據 CloudWatch 中的指標設定警示。

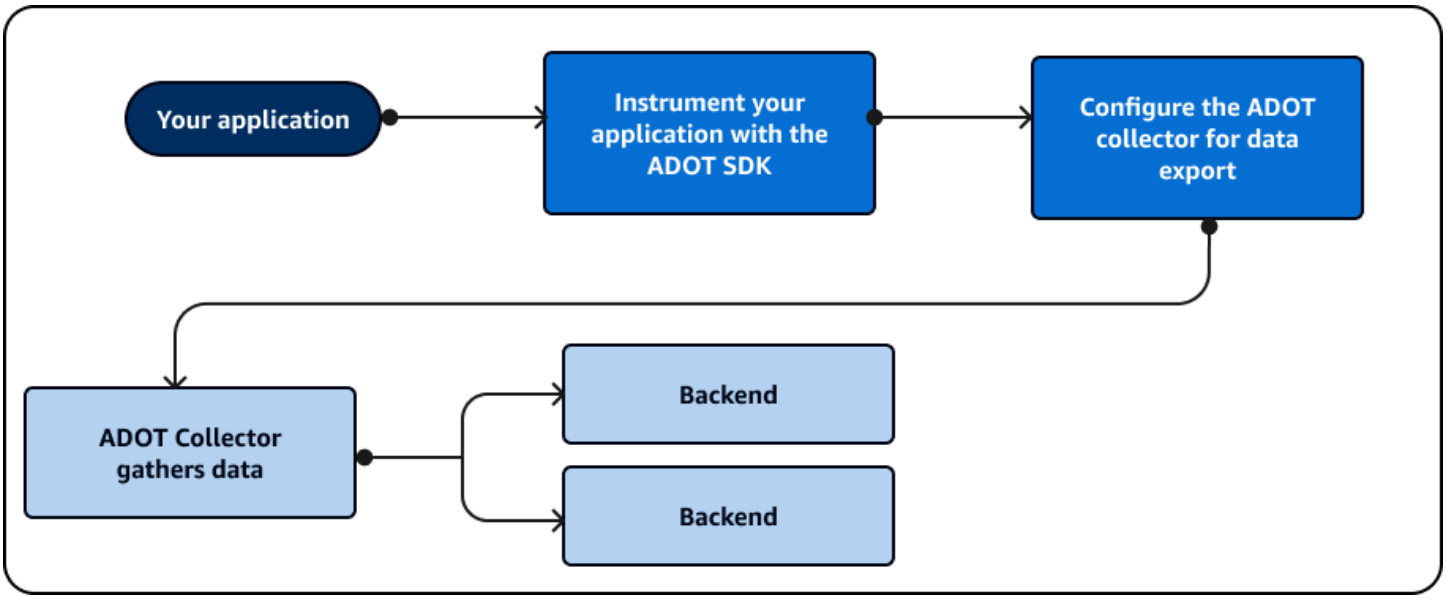
如果您使用 ADOT SDK，則有下列選項，並搭配 代理程式：

- 搭配 CloudWatch 代理程式使用 ADOT SDK – 建議使用。
- 搭配 ADOT Collector 使用 ADOT SDK – 如果您想要搭配 AWS 多層安全性和最佳化使用與廠商無關的軟體，建議使用 。

若要使用 ADOT SDK，請執行下列動作：

- 使用 ADOT SDK 檢測您的應用程式。如需詳細資訊，請參閱 ADOT 技術文件中程式設計語言的文件。
- 設定ADOT收集器以告知收集資料的位置。

ADOT 收集器收到您的資料後，會將其傳送至您在ADOT組態中指定的後端。 ADOT可以將資料傳送至多個後端，包括 以外的廠商 AWS，如下圖所示：



AWS 會定期更新ADOT以新增功能，並與 OpenTelemetry 架構保持一致。開發的更新和未來計劃ADOT是可供公眾使用的藍圖的一部分。 ADOT支援多種程式設計語言，包括下列項目：

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

如果您使用的是 Python， ADOT可以自動檢測您的應用程式。若要開始使用 ADOT，請參閱 Distro for OpenTelemetry Collector 簡介和入門。 AWS OpenTelemetry

使用 X-Ray SDK

X-Ray SDK 是一組 AWS APIs和程式庫，可將 AWS 資料傳送至後端服務。使用 X-Ray 開發套件來檢測您的應用程式並收集追蹤資料。您無法使用 X-Ray 開發套件來收集日誌或指標資料。

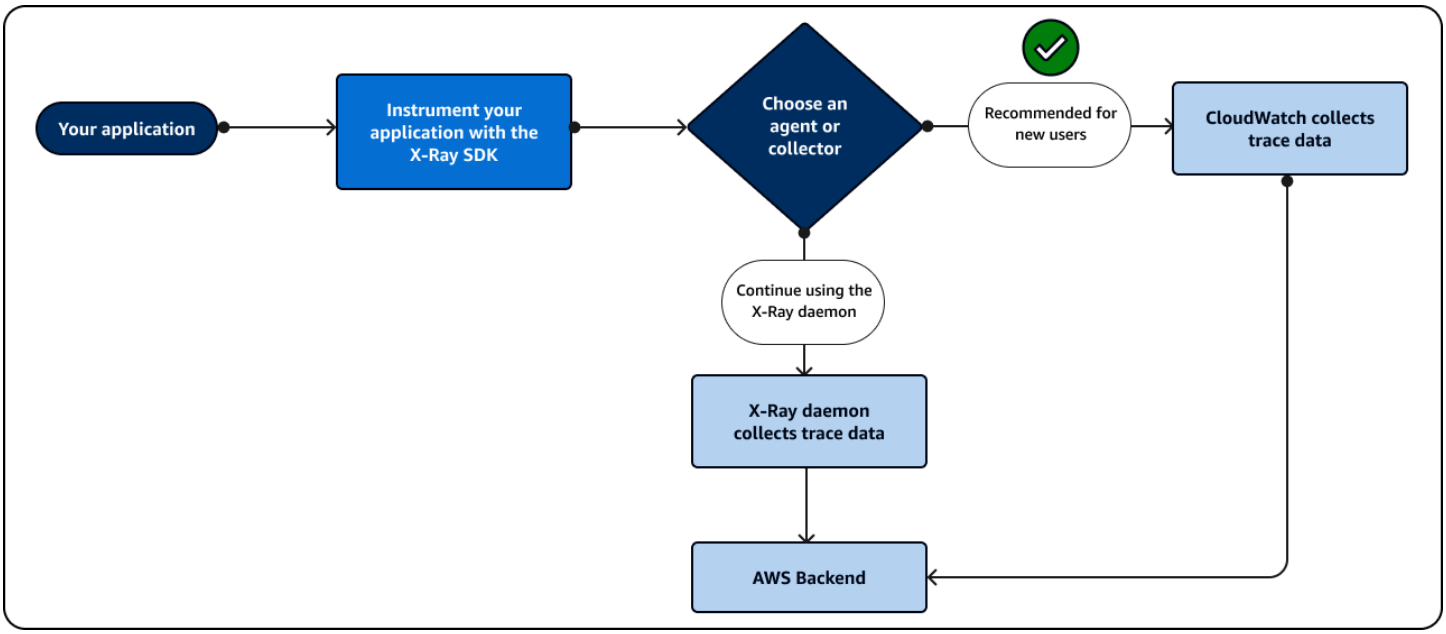
如果您使用的是 X-Ray 開發套件，您有下列選項，並搭配 代理程式：

- 搭配 使用 X-Ray 開發套件 AWS X-Ray 協助程式 – 如果您不想更新應用程式碼，請使用此選項。
- 搭配 CloudWatch 代理程式使用 X-Ray 開發套件 –（建議） CloudWatch 代理程式與 X-Ray 開發套件相容。

若要使用 X-Ray 開發套件，請執行下列動作：

- 使用 X-Ray SDK 檢測您的應用程式。
- 設定收集器以告知收集資料的位置。您可以使用 CloudWatch 代理程式或 X-Ray 協助程式來收集追蹤資訊。

收集器或代理程式收到您的資料後，會將其傳送至您在代理程式組態中指定的 AWS 後端。X-Ray SDK 只能將資料傳送至 AWS 後端，如下圖所示：



如果您使用的是 Java，您可以使用 X-Ray 開發套件自動檢測您的應用程式。若要開始使用 X-Ray 開發套件，請參閱與下列程式設計語言相關聯的程式庫：

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

使用主控台

如果您想要圖形化使用者介面 (GUI) 需要最少的編碼，請使用 主控台。初次使用 X-Ray 的使用者可以使用預先建置的視覺化效果快速開始，並執行基本任務。您可以直接從主控台執行下列動作：

- 啟用 X-Ray。
- 檢視應用程式效能的高階摘要。
- 檢查應用程式的運作狀態。
- 識別高階錯誤。
- 檢視基本追蹤摘要。

您可以使用 Amazon CloudWatch 主控台，網址為 https://console.aws.amazon.com/cloudwatch/：// Amazon CloudWatch 主控台，或 X-Ray 主控台，網址為 https://console.aws.amazon.com/xray/home：//www.microsoft.com，與 X-Ray 互動。

使用 Amazon CloudWatch 主控台

CloudWatch 主控台包含從 X-Ray 主控台重新設計的新 X-Ray 功能，讓您更輕鬆地使用。如果您使用 CloudWatch 主控台，您可以檢視 CloudWatch 日誌和指標以及 X-Ray 追蹤資料。使用 CloudWatch 主控台來檢視和分析資料，包括下列項目：

- X-Ray 追蹤 – 檢視、分析和篩選與您的應用程式相關聯的追蹤，因為它可處理請求。使用這些追蹤來尋找高延遲、偵錯錯誤，以及最佳化您的應用程式工作流程。檢視追蹤地圖和服務地圖，以查看應用程式工作流程的視覺化呈現。
- 日誌 – 檢視、分析和篩選應用程式產生的日誌。使用日誌對錯誤進行故障診斷，並根據特定日誌值設定監控。
- 指標 – 使用資源發出的指標或建立您自己的指標，測量和監控您的應用程式效能。在圖表中檢視這些指標。
- 監控網路和基礎設施 – 監控主要網路是否有中斷，以及基礎設施的運作狀態和效能，包括容器化應用程式、 AWS 其他服務和用戶端。
- 下列使用 X-Ray 主控台區段中列出的 X-Ray 主控台的所有功能。

如需 CloudWatch 主控台的詳細資訊，請參閱 Amazon CloudWatch 入門。

登入 Amazon CloudWatch 主控台，網址為 https://console.aws.amazon.com/cloudwatch/：//https：// Amazon CloudWatch 主控台。

使用 X-Ray 主控台

X-Ray 主控台提供應用程式請求的分散式追蹤。如果您想要更簡單的主控台體驗或不想更新您的應用程式碼，請使用 X-Ray 主控台。 AWS 不再開發 X-Ray 主控台。X-Ray 主控台包含下列適用於經檢測應用程式的功能：

- 洞見 – 自動偵測應用程式效能中的異常，並找出根本原因。Insights 包含在 CloudWatch 主控台的 Insights 下。如需詳細資訊，請參閱 中的使用 X-Ray Insights使用 X-Ray 主控台。
- 服務映射 – 檢視應用程式的圖形結構及其與用戶端、資源、服務和相依性的連線。
- 追蹤 – 查看應用程式在處理請求時所產生的追蹤概觀。使用追蹤資料來了解您的應用程式如何針對基本指標執行，包括HTTP回應和回應時間。
- 分析 – 使用回應時間分佈的圖形來解譯、探索和分析追蹤資料。
- 組態 – 建立自訂追蹤以變更下列項目的預設組態：
  - 取樣 – 建立規則，定義取樣應用程式以取得追蹤資訊的頻率。如需詳細資訊，請參閱在 使用 X-Ray 主控台 中設定抽樣規則。
  - 加密 – 使用您可以使用 稽核或停用的金鑰來加密靜態資料 AWS Key Management Service。
  - 群組 – 使用篩選條件表達式來定義具有常見功能的追蹤群組，例如 URL 的名稱或回應時間。如需詳細資訊，請參閱設定群組。

登入 X-Ray 主控台，網址為 https：//https://console.aws.amazon.com/xray/home。

探索 X-Ray 主控台

使用 X-Ray 主控台檢視應用程式提供的請求的服務映射和相關追蹤，以及設定群組和取樣規則，這些規則會影響追蹤傳送至 X-Ray 的方式。

> **Note**
>
> X-Ray Service 映射和 CloudWatch ServiceLens 映射已合併到 Amazon CloudWatch 主控台中的 X-Ray 追蹤映射中。開啟 CloudWatch 主控台，然後從左側導覽窗格選擇 X-Ray 追蹤下的追蹤映射。
>
> CloudWatch 現在包含 Application Signals，可探索和監控您的應用程式服務、用戶端、Synthetics Canary 和服務相依性。使用 Application Signals 查看服務清單或視覺化地圖，根據您的服務等級目標 (SLO) 檢視運作狀態指標，並深入了解相關的 X-Ray 追蹤以取得更詳細的疑難排解。

主要 X-Ray 主控台頁面是追蹤映射，這是 X-Ray 從應用程式產生的追蹤資料產生的 JSON 服務圖形視覺化表示。映射由您帳戶中每個處理請求的應用程式服務節點、代表請求來源的上游用戶端節點，以及代表處理請求過程中應用程式所使用 web 服務及資源的下游服務節點組成。還有其他頁面可檢視追蹤和追蹤詳細資訊，以及設定群組和取樣規則。

檢視 X-Ray 的主控台體驗，並與以下各節中的 CloudWatch 主控台進行比較。

探索 X-Ray 和 CloudWatch 主控台

- 使用 X-Ray 追蹤映射
- 檢視追蹤和追蹤詳細資訊
- 使用篩選條件表達式
- 跨帳戶追蹤
- 追蹤事件驅動型應用程式
- 使用延遲長條圖
- 使用 X-Ray 洞察
- 與 Analytics 主控台互動
- 設定 群組
- 設定 取樣規則
- 主控台深層連結

使用 X-Ray 追蹤映射

檢視 X-Ray 追蹤映射，以識別發生錯誤的服務、具有高延遲的連線，或失敗請求的追蹤。

> **Note**
>
> CloudWatch 現在包含 Application Signals，可探索和監控您的應用程式服務、用戶端、合成 Canary 和服務相依性。使用 Application Signals 查看服務清單或視覺化地圖，根據您的服務等級目標 (SLO) 檢視運作狀態指標，並深入了解相關的 X-Ray 追蹤以取得更詳細的疑難排解。X-Ray 服務映射和 CloudWatch ServiceLens 映射會合併到 Amazon CloudWatch 主控台內的 X-Ray 追蹤映射中。開啟 CloudWatch 主控台，然後從左側導覽窗格選擇 X-Ray 追蹤下的追蹤地圖。

檢視追蹤映射

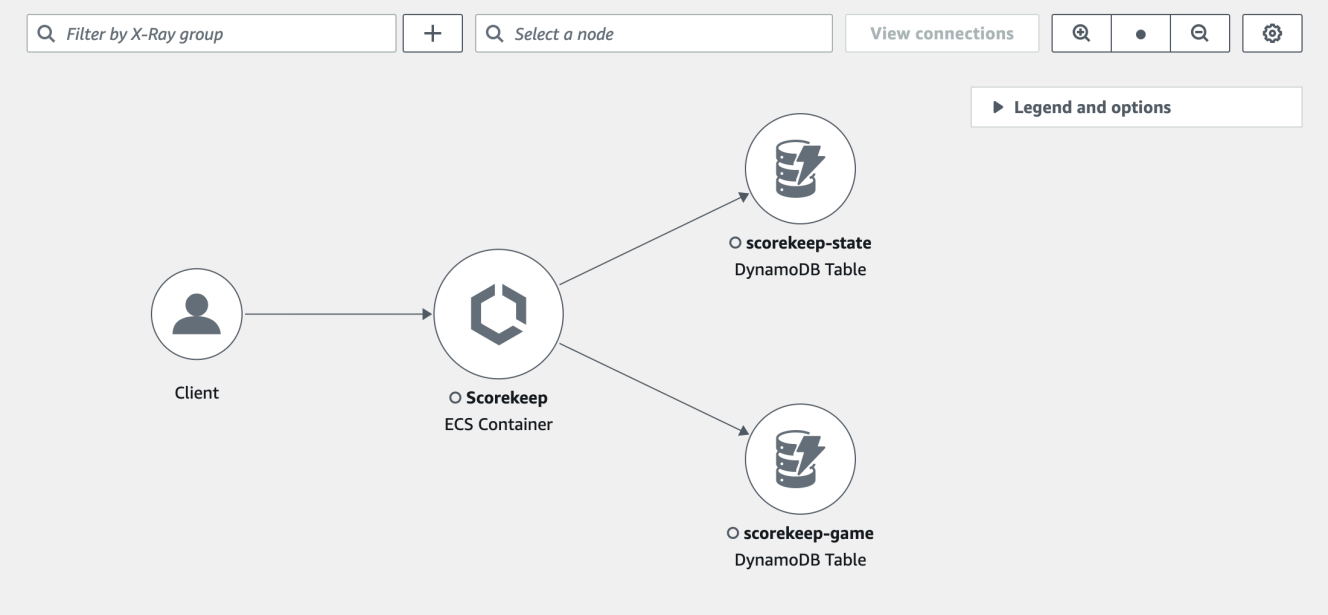
追蹤映射是應用程式所產生追蹤資料的視覺化呈現。地圖會顯示服務節點，其可處理請求、代表請求來源的上游用戶端節點，以及代表應用程式在處理請求時所使用的 Web 服務和資源的下游服務節點。

追蹤映射會顯示使用 Amazon SQS 和 Lambda 之事件驅動應用程式的追蹤連線檢視。如需詳細資訊，請參閱追蹤事件驅動的應用程式。追蹤映射也支援跨帳戶追蹤，在單一映射中顯示來自多個帳戶的節點。

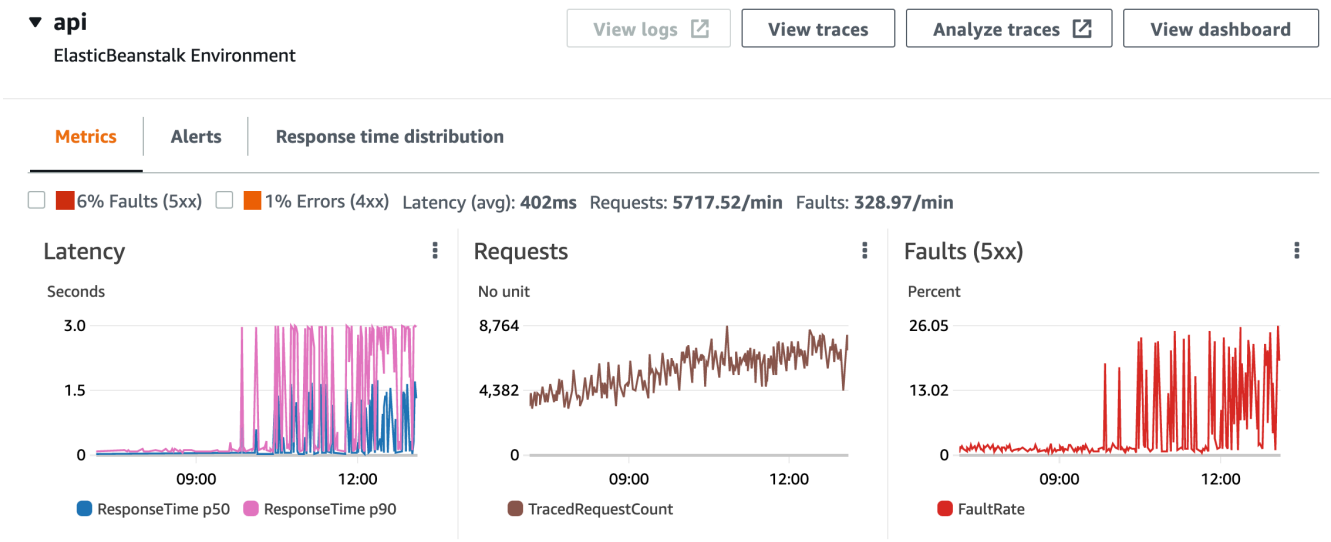
CloudWatch console

在 CloudWatch 主控台中檢視追蹤映射

1. 開啟 CloudWatch 主控台。在左側導覽窗格中的 X-Ray 追蹤區段下，選擇追蹤地圖。



2. 選擇服務節點來檢視該節點的請求，或是兩個節點間的邊緣來檢視在該連線上進行的請求。
3. 其他資訊會顯示在追蹤映射下方，包括指標、提醒和回應時間分佈的標籤。在指標索引標籤上，選取每個圖形內的範圍來向下切入以檢視更多詳細資訊，或選擇錯誤或錯誤選項來篩選追蹤。在回應時間分佈索引標籤上，選取圖形內的範圍，以依回應時間篩選追蹤。



4. 選擇檢視追蹤來檢視追蹤，或者如果已套用篩選條件，請選擇檢視篩選的追蹤。
5. 選擇檢視日誌以查看與所選節點相關聯的 CloudWatch 日誌。並非所有追蹤映射節點都支援檢視日誌。如需詳細資訊，請參閱對 CloudWatch 日誌進行故障診斷。

追蹤映射會以顏色概述每個節點內的問題：

- 紅色表示伺服器故障 (500 系列錯誤)
- 黃色表示用戶端錯誤 (400 系列錯誤)
- 紫色表示調節錯誤 (429 請求數太多)

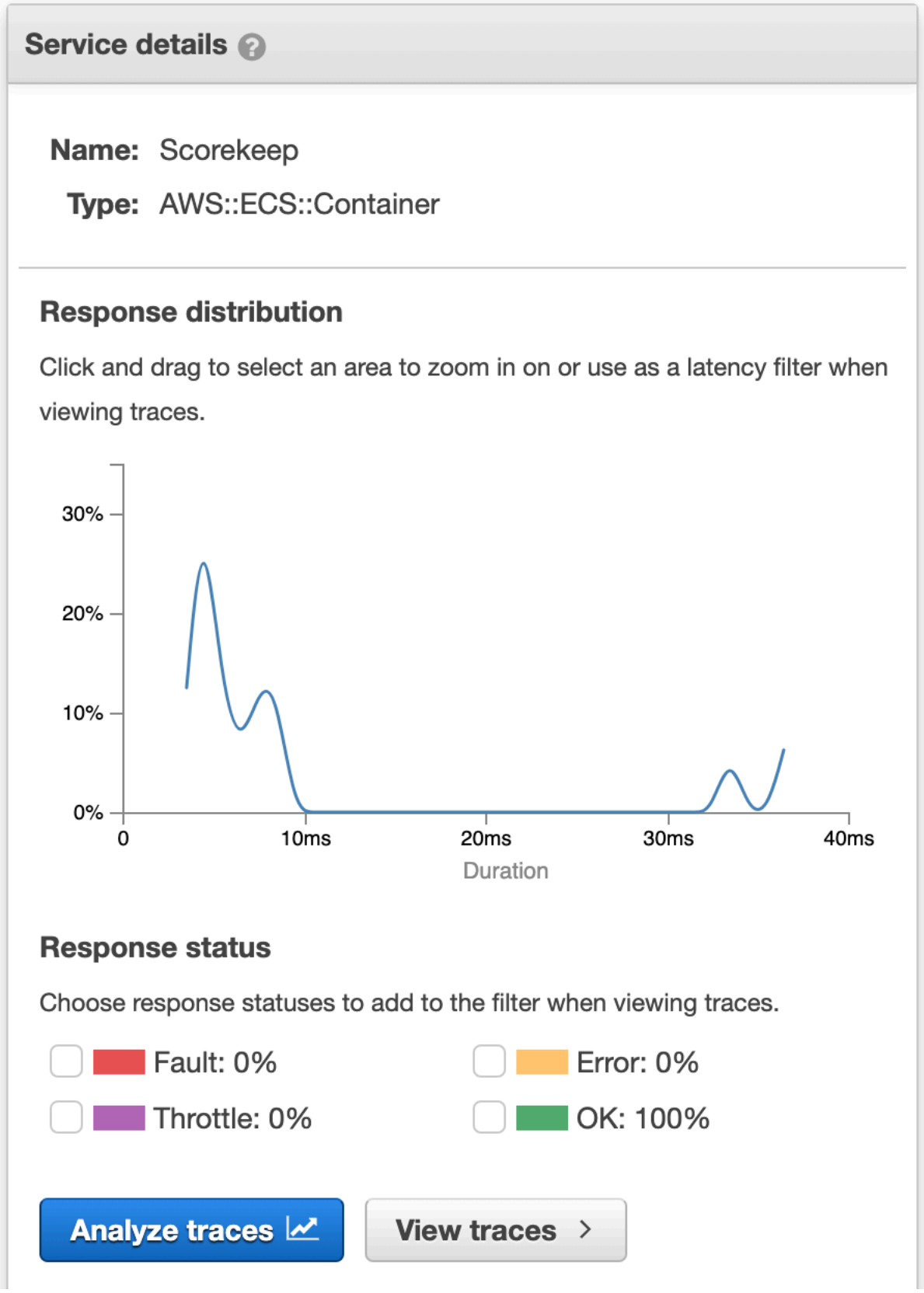
如果您的追蹤映射很大，請使用螢幕控制項或滑鼠來放大和縮小和移動映射。

X-Ray console

檢視服務映射

1. 開啟 X-Ray 主控台。預設會顯示服務映射。您也可以從左側導覽窗格中選擇服務地圖。

2. 選擇服務節點來檢視該節點的請求，或是兩個節點間的邊緣來檢視在該連線上進行的請求。
3. 使用回應分佈長條圖依持續時間篩選追蹤，然後選取您要檢視追蹤的狀態碼。然後選擇 View traces (檢視追蹤) 來使用套用的篩選條件表達式開啟追蹤清單。


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

服務映射會透過根據成功呼叫與錯誤及故障的比例，將每個節點標上顏色，來指出每個節點的運作狀態。

- 綠色表示成功呼叫
- 紅色表示伺服器故障 (500 系列錯誤)
- 黃色表示用戶端錯誤 (400 系列錯誤)
- 紫色表示調節錯誤 (429 請求數太多)

如果您的服務映射很大，請使用螢幕控制項或滑鼠來放大和縮小和移動映射。

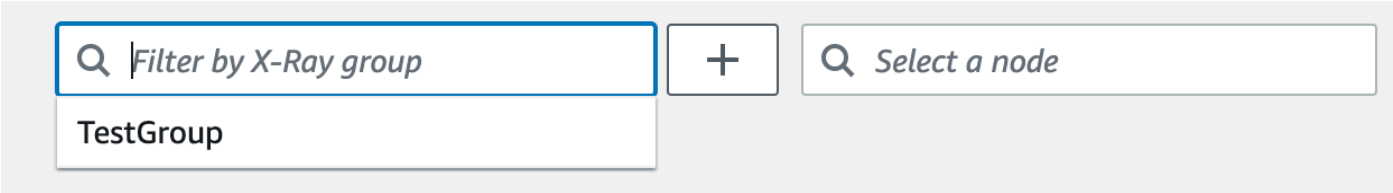
> **Note**
>
> X-Ray 追蹤映射最多可顯示 10，000 個節點。在服務節點總數超過此限制的罕見情況下，您可能會收到錯誤，無法在主控台中顯示完整的追蹤映射。

依群組篩選追蹤映射

使用篩選條件表達式，您可以定義要在群組中包含追蹤的條件。使用下列步驟在追蹤映射中顯示該特定群組。

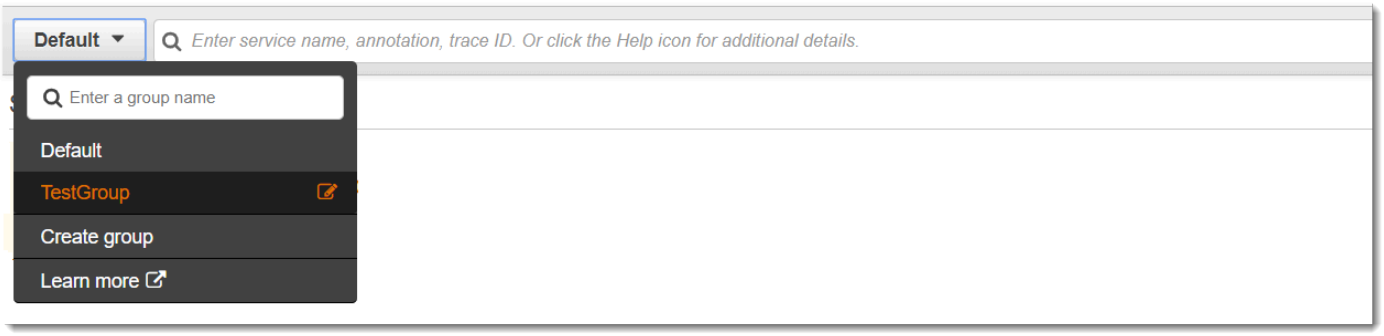
CloudWatch console

從追蹤映射左上角的群組篩選條件中選擇群組名稱。



X-Ray console

從下拉式功能表左側的搜尋列，選擇群組名稱。



現在將篩選服務映射，以顯示符合所選群組篩選條件表達式的追蹤。

追蹤地圖圖例和選項

追蹤映射包含圖例和數個選項，用於自訂映射顯示。

CloudWatch console

選擇地圖右上角的圖例和選項下拉式清單。選擇節點中顯示的內容，包括：

- 指標會顯示平均回應時間和在所選時間範圍內每分鐘傳送的追蹤數量。
- 節點會顯示每個節點內的服務圖示。

從偏好設定窗格選擇其他地圖設定，可透過地圖右上角的齒輪圖示存取。這些設定包括選取用於判斷每個節點大小的指標，以及哪些 Canary 應顯示在地圖上。

X-Ray console

選擇地圖右上角的地圖圖例連結，以顯示服務地圖圖例。您可以在追蹤映射的右下角選擇服務映射選項，包括：

- 服務圖示會切換每個節點內顯示的內容，顯示服務圖示，或在所選時間範圍內每分鐘傳送的平均回應時間和追蹤數量。
- 節點大小：無會將所有節點設為相同大小。
- 節點大小調整：運作狀態會依受影響的請求數量來調整節點大小，包括錯誤、故障或限流請求。
- 節點大小：流量會依請求總數來調整節點大小。

檢視追蹤和追蹤詳細資訊

使用 X-Ray 主控台中的追蹤頁面，依 URL、回應碼或追蹤摘要中的其他資料尋找追蹤。從追蹤清單中選取追蹤之後，追蹤詳細資訊頁面會顯示與所選追蹤相關聯的服務節點映射，以及追蹤區段的時間軸。

檢視追蹤

CloudWatch console

在 CloudWatch 主控台中檢視追蹤

1. 登入 AWS Management Console ，並在 https：//https://console.aws.amazon.com/cloudwatch/ 開啟 CloudWatch 主控台。
2. 在左側導覽窗格中，選擇 X-Ray 追蹤，然後選擇追蹤。您可以依群組篩選或輸入篩選條件表達式。這會篩選頁面底部的追蹤區段中顯示的追蹤。

   或者，您可以使用服務映射導覽至特定服務節點，然後檢視追蹤。這會開啟已套用查詢的追蹤頁面。
3. 在查詢精簡器區段中精簡您的查詢。若要依常見屬性篩選追蹤，請從精簡查詢依據旁的向下箭頭選擇一個選項。選項包括下列項目：

   - 節點 – 依服務節點篩選追蹤。
   - 資源 ARN – 依與追蹤相關聯的資源篩選追蹤。這些資源的範例包括 Amazon Elastic Compute Cloud (Amazon EC2) 執行個體、 AWS Lambda 函數或 Amazon DynamoDB 資料表。
   - 使用者 – 使用使用者 ID 篩選追蹤。
   - 錯誤根本原因訊息 – 依錯誤根本原因篩選追蹤。
   - URL – 依應用程式使用的 URL 路徑篩選追蹤。
   - HTTP 狀態碼 – 依應用程式傳回的 HTTP 狀態碼篩選追蹤。您可以指定自訂回應代碼，或從下列項目中選取：
     - `200` – 請求成功。
     - `401` – 請求缺少有效的身分驗證登入資料。
     - `403` – 請求缺少有效許可。
     - `404` – 伺服器找不到請求的資源。
     - `500` – 伺服器遇到未預期的條件並產生內部錯誤。

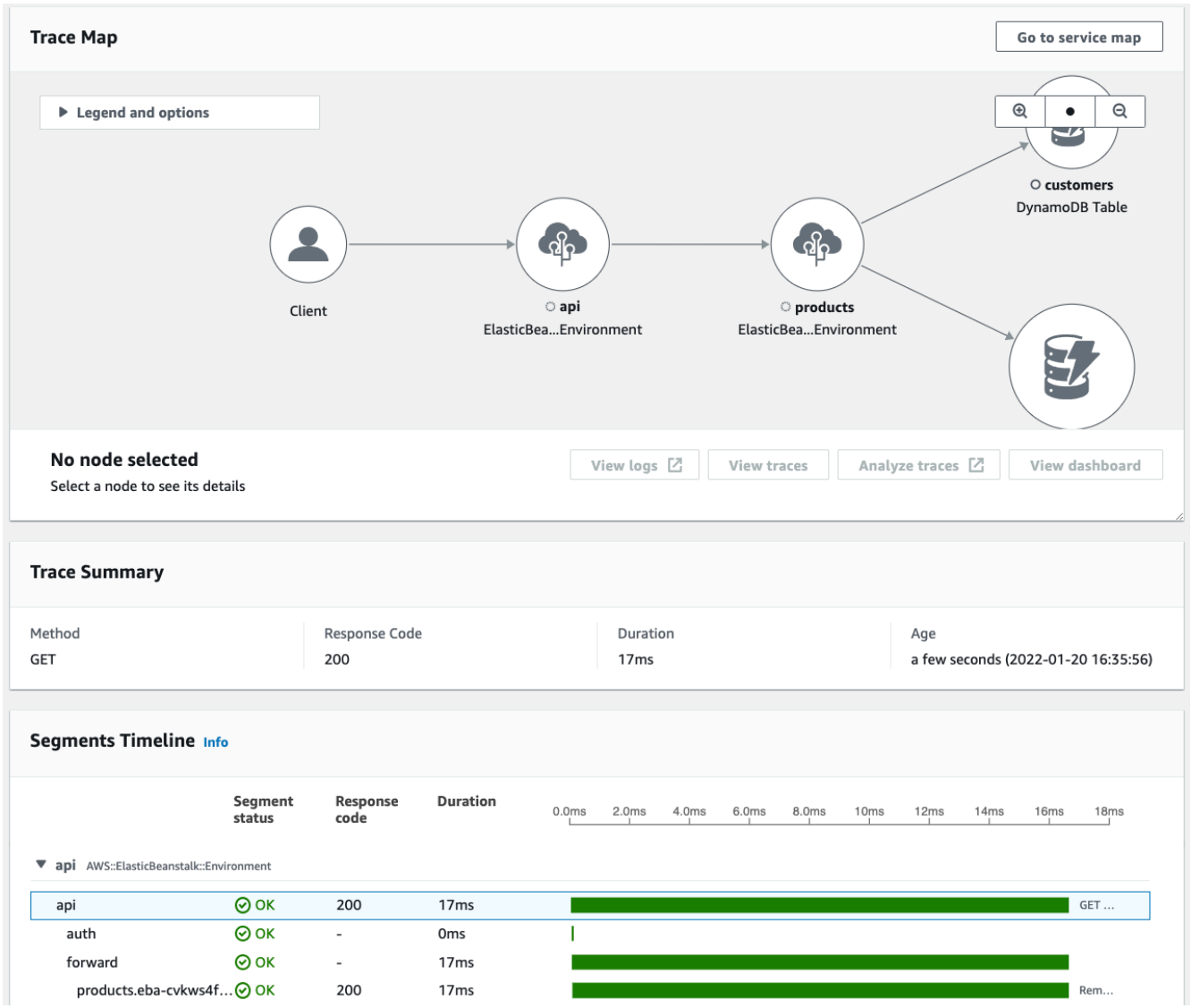
選擇一或多個項目，然後選擇新增至查詢，以新增至頁面頂端的篩選條件表達式。

4. 若要尋找單一追蹤，請直接在查詢欄位中輸入追蹤 ID。您可以使用 X-Ray 格式或全球資訊網協會 (W3C) 格式。例如，使用 AWS Distro for OpenTelemetry 建立的追蹤是 W3C 格式。

   > **Note**
   >
   > 當您查詢使用 W3C-format追蹤 ID 建立的追蹤時，主控台會以 X-Ray 格式顯示相符的追蹤。例如，如果您`4efaaf4d1e8720b39541901950019ee5`以 W3C 格式查詢 ，主控台會顯示 X-Ray 對等項目：`1-4efaaf4d-1e8720b39541901950019ee5`。

5. 選擇隨時執行查詢，以在頁面底部的追蹤區段中顯示相符的追蹤清單。
6. 若要顯示單一追蹤的追蹤詳細資訊頁面，請從清單中選擇追蹤 ID。

   下圖顯示追蹤映射，其中包含與追蹤相關聯的服務節點，以及節點之間的邊緣，代表由構成追蹤的區段所採取的路徑。追蹤摘要遵循追蹤地圖。摘要包含範例`GET`操作、其回應碼、追蹤執行所花費的持續時間，以及請求的存留期的相關資訊。客群時間軸遵循追蹤摘要，顯示追蹤客群和子客群的持續時間。



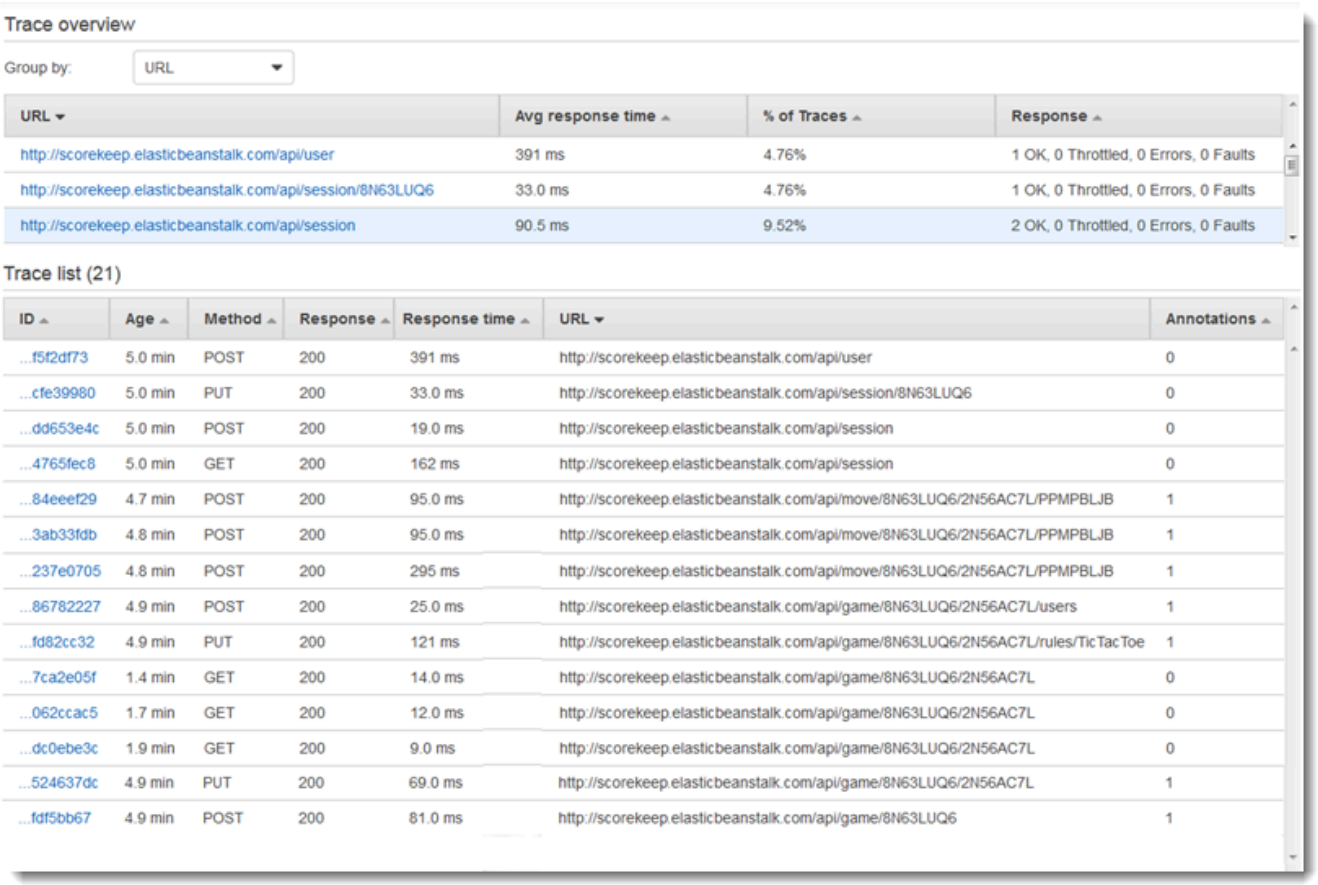
如果您有使用 Amazon SQS 和 Lambda 的事件驅動應用程式，您可以在追蹤映射中查看每個請求的追蹤連線檢視。在地圖中，來自訊息生產者的追蹤會連結到來自 AWS Lambda 消費者的追蹤，並以虛線邊緣顯示。如需事件驅動型應用程式的詳細資訊，請參閱 追蹤事件驅動型應用程式。

追蹤和追蹤詳細資訊頁面也支援跨帳戶追蹤，可在追蹤清單中和單一追蹤映射內列出來自多個帳戶的追蹤。

X-Ray console

在 X-Ray 主控台中檢視追蹤

1. 在 X-Ray 主控台中開啟追蹤頁面。追蹤概觀面板會顯示依常見功能分組的追蹤清單，包括錯誤根本原因、ResourceARN 和 InstanceId。
2. 若要選取常見功能以檢視一組分組的追蹤，請展開分組依據旁的向下箭頭。下圖顯示依 URL 分組的追蹤概觀AWS X-Ray 範例應用程式，以及相關聯的追蹤清單。

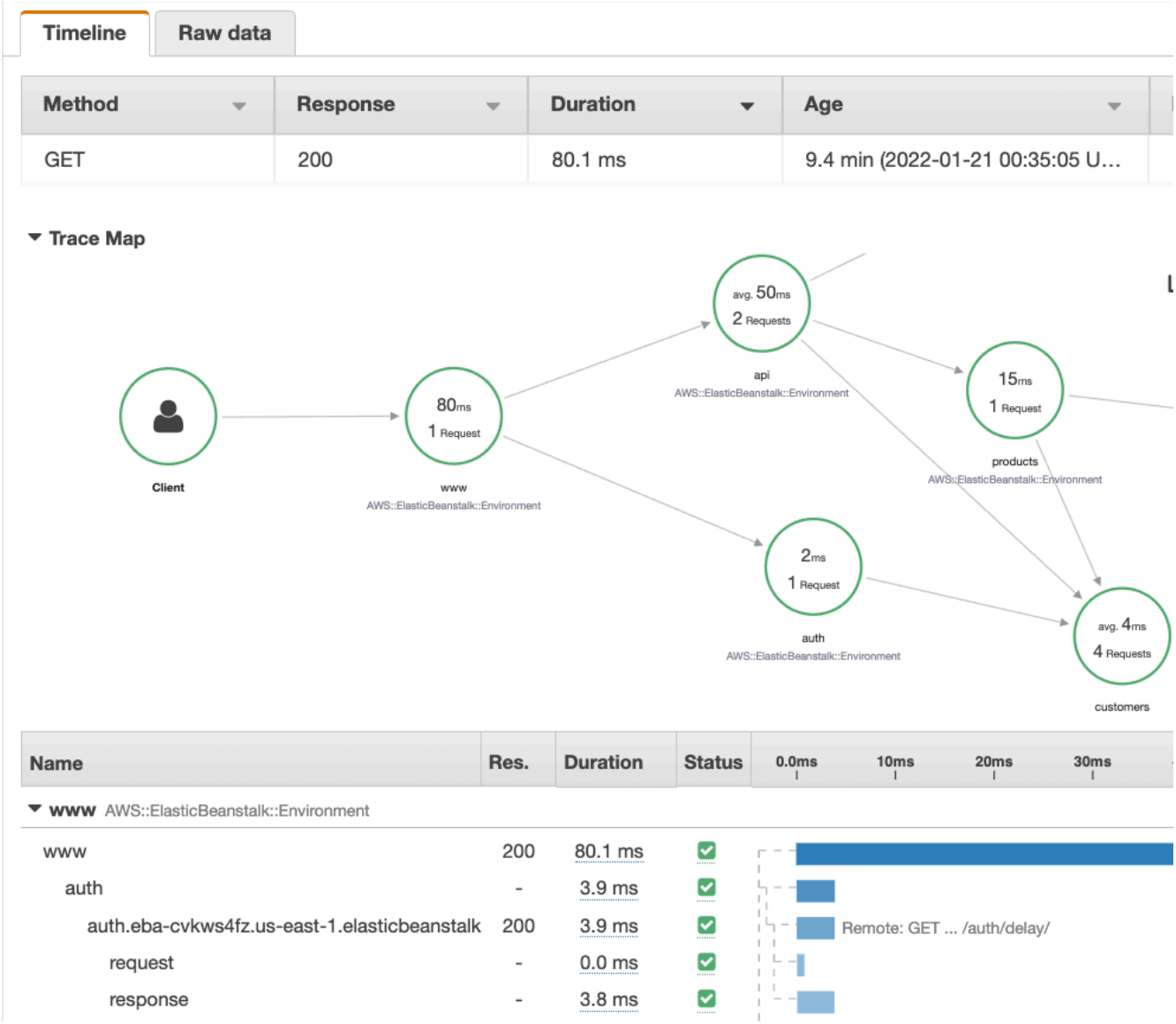


3. 選擇追蹤的 ID，以在追蹤清單下檢視。您也可以在導覽窗格中選擇服務映射，以檢視特定服務節點的追蹤。然後，您可以檢視與該節點相關聯的追蹤。

   時間軸索引標籤會顯示追蹤的請求流程，並包含下列項目：

   - 追蹤中每個區段的路徑映射。
   - 區段到達追蹤映射中節點所需的時間。
   - 對追蹤映射中的節點提出了多少請求。

下圖顯示與對範例應用程式提出之GET請求相關聯的追蹤地圖範例。箭頭顯示每個區段完成請求所採取的路徑。服務節點會顯示請求期間提出的GET請求數量。



如需時間軸索引標籤的詳細資訊，請參閱下列探索追蹤時間軸區段。

原始資料索引標籤會以 JSON 格式顯示追蹤的相關資訊，以及構成追蹤的區段和子區段。此資訊可能包括下列項目：

- 時間戳記
- 唯一 ID
- 與區段或子區段相關聯的資源

- 區段或子區段的來源或原始伺服器
- 有關對應用程式請求的其他資訊，例如來自 HTTP 請求的回應

探索追蹤時間軸

時間軸區段會在水平長條旁顯示區段和子區段的階層，對應於他們用來完成任務的時間。清單中的第一個項目是區段，代表服務為單一請求記錄的所有資料。子區段會縮排，並列在區段後面。資料欄包含每個區段的相關資訊。

CloudWatch console

在 CloudWatch 主控台中，客群時間軸會提供下列資訊：

- 第一欄：列出所選追蹤中的區段和子區段。
- 區段狀態欄：列出每個區段和子區段的狀態結果。
- 回應碼欄：列出區段或子區段提出的瀏覽器請求的 HTTP 回應狀態碼，當可用時。
- 持續時間欄：列出區段或子區段執行的時間長度。
- 託管於資料欄：列出執行區段或子區段的命名空間或環境，如適用。如需詳細資訊，請參閱收集的維度和維度組合。
- 最後一欄：顯示與時間軸中其他區段或子區段相關的區段或子區段執行持續時間對應的水平長條。

若要依服務節點分組區段和子區段清單，請開啟依節點分組。

X-Ray console

在追蹤詳細資訊頁面中，選擇時間軸索引標籤，以查看構成追蹤的每個區段和子區段的時間軸。

在 X-Ray 主控台中，時間軸會提供下列資訊：

- 名稱欄：列出追蹤中區段和子區段的名稱。
- Res. 欄：列出區段或子區段提出的瀏覽器請求的 HTTP 回應狀態碼，當可用時。
- 持續時間欄：列出區段或子區段執行的時間長度。
- 狀態欄：列出區段或子區段狀態的結果。
- 最後一欄：顯示與時間軸中其他區段或子區段相關的區段或子區段執行持續時間對應的水平長條。

若要查看主控台用來產生時間軸的原始追蹤資料，請選擇原始資料索引標籤。原始資料會顯示追蹤的相關資訊，以及以 JSON 格式編寫追蹤的區段和子區段。此資訊可能包括下列項目：

- 時間戳記
- 唯一 ID
- 與區段或子區段相關聯的資源
- 區段或子區段的來源或原始伺服器
- 有關對應用程式提出請求的其他資訊，例如來自 HTTP 請求的回應。

當您使用經檢測的 AWS 開發套件HTTP、 或 SQL用戶端呼叫外部資源時，X-Ray 開發套件會自動記錄子區段。您也可以使用 X-Ray SDK 記錄任何函數或程式碼區塊的自訂子區段。自訂子區段開啟時記錄的其他子區段會成為自訂子區段的子區段。

檢視區段詳細資訊

從追蹤時間軸中，選擇要檢視其詳細資訊的客群名稱。

區段詳細資訊面板會顯示概觀、資源、註釋、中繼資料、例外狀況和 SQL 索引標籤。適用下列情況：

- Overview (概觀) 標籤會顯示請求及回應的相關資訊。資訊包括名稱、開始時間、結束時間、持續時間、請求 URL、請求操作、請求回應碼，以及任何錯誤和故障。
- 區段的資源索引標籤會顯示 X-Ray 開發套件的資訊，以及執行您應用程式 AWS 的資源。使用 X-Ray SDK 的 Amazon EC2 AWS Elastic Beanstalk或 Amazon ECS 外掛程式來記錄服務特定的資源資訊。如需外掛程式的詳細資訊，請參閱《》中的服務外掛程式一節設定適用於 Java 的 X-Ray 開發套件。
- 其餘索引標籤會顯示針對區段記錄的註釋、中繼資料和例外狀況。從檢測的請求產生例外狀況時，系統會自動擷取例外狀況。註釋和中繼資料包含您使用 X-Ray SDK 提供的操作記錄的其他資訊。若要將註釋或中繼資料新增至您的客群，請使用 X-Ray 開發套件。如需詳細資訊，請參閱 中使用 AWS X-Ray SDKs 檢測應用程式下列出的特定語言連結檢測您的應用程式 AWS X-Ray。

檢視子區段詳細資訊

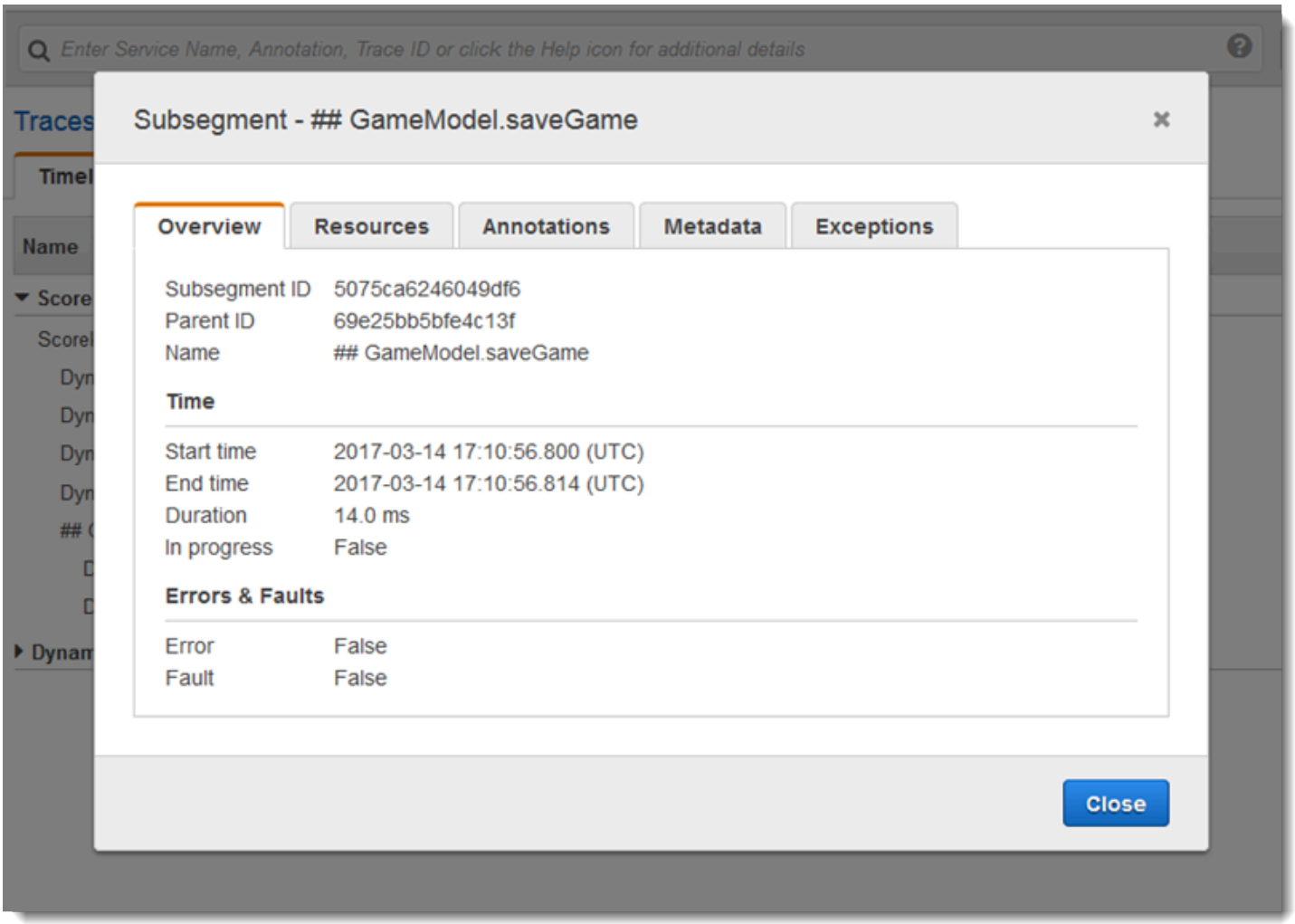
從追蹤時間軸中，選擇子區段的名稱以檢視其詳細資訊：

- 概觀索引標籤包含請求和回應的相關資訊。這包括名稱、開始時間、結束時間、持續時間、請求 URL、請求操作、請求回應碼，以及任何錯誤和故障。針對使用受檢測用戶端產生的子區段，Overview (概觀) 標籤包含從您應用程式觀點的請求和回應相關資訊。

- 子區段的資源索引標籤會顯示用來執行子區段之 AWS 資源的詳細資訊。例如，資源索引標籤可能包含 AWS Lambda 函數 ARN、DynamoDB 資料表的相關資訊、呼叫的任何操作，以及請求 ID。
- 其餘索引標籤會顯示子區段上記錄的註釋、中繼資料和例外狀況。從檢測的請求產生例外狀況時，系統會自動擷取例外狀況。註釋和中繼資料包含您使用 X-Ray SDK 提供的操作記錄的其他資訊。使用 X-Ray SDK 將註釋或中繼資料新增至您的客群。如需詳細資訊，請參閱 中使用 SDKs 檢測應用程式 AWS X-Ray 下列出的特定語言連結檢測您的應用程式 AWS X-Ray。

針對自訂子區段，Overview (概觀) 標籤會顯示子區段的名稱，您可以設定該名稱來指定其記錄的程式碼或函數區域。如需詳細資訊，請參閱 中使用 AWS X-Ray SDKs 檢測應用程式下列出的特定語言連結使用適用於 Java 的 X-Ray 開發套件產生自訂子區段。

下圖顯示自訂子區段的概觀索引標籤。概觀包含子區段 ID、父系 ID、名稱、開始和結束時間、持續時間、狀態和錯誤或故障。



自訂子區段的中繼資料索引標籤包含該子區段所用資源的JSON格式資訊。

使用篩選條件表達式

使用篩選條件表達式來檢視特定請求、服務、兩個服務（邊緣）之間的連線，或滿足條件之請求的追蹤映射或追蹤。X-Ray 提供篩選表達式語言，根據請求標頭中的資料、回應狀態和原始區段上的索引欄位來篩選請求、服務和邊緣。

當您選擇要在 X-Ray 主控台中檢視的追蹤期間時，可能會得到比主控台可顯示更多的結果。主控台會在右上角顯示已掃描的追蹤數目，但實際上可用的追蹤可能更多。您可以使用篩選條件表達式，將結果縮小為只您想要尋找的追蹤。

主題

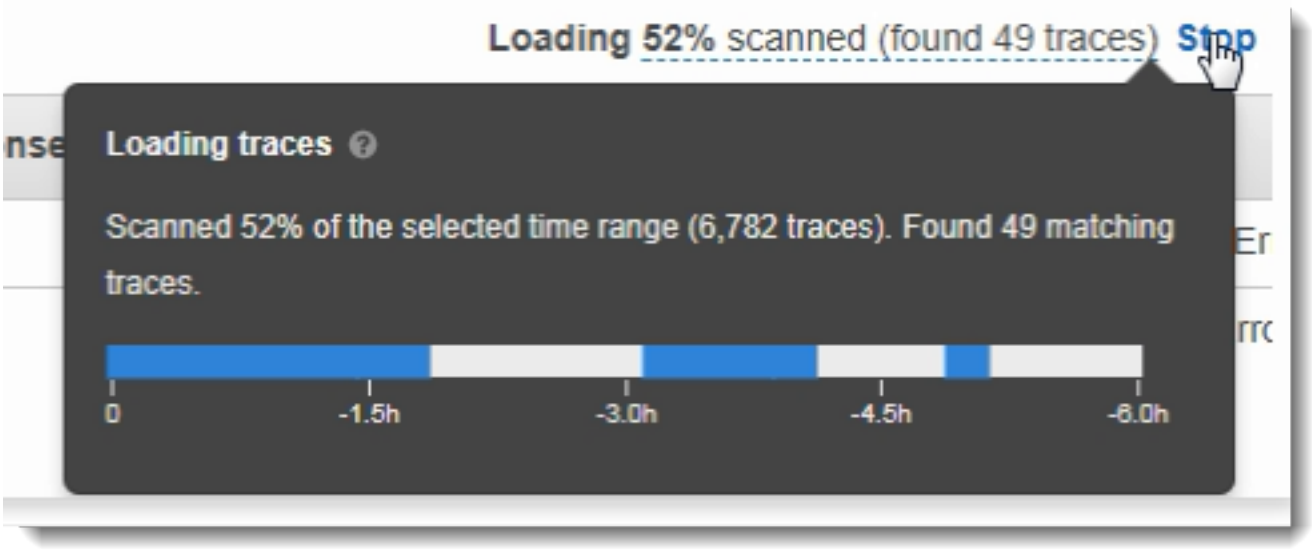
篩選條件表達式詳細資訊

當您在追蹤映射中選擇節點時，主控台會根據節點的服務名稱，以及根據您的選擇存在的錯誤類型，建構篩選條件表達式。若要尋找顯示效能問題或與特定請求相關的追蹤，您可以調整主控台提供的表達式，或是自行建立。如果您使用 X-Ray SDK 新增註釋，您也可以根據註釋索引鍵的存在或索引鍵的值進行篩選。

> **Note**
>
> 如果您在追蹤映射中選擇相對時間範圍並選擇節點，主控台會將時間範圍轉換為絕對開始和結束時間。為了確保搜尋結果顯示節點的追蹤，並避免掃描未作用中節點的時間，時間範圍只包含節點傳送追蹤的時間。若要搜尋相對於目前的時間，您可以在追蹤頁面切換回相對時間範圍，並再次掃描。

如果可用結果超過主控台可以顯示的範圍，主控台會顯示相符的追蹤數和已掃描的追蹤數。顯示的百分比為所選時間範圍內已掃描的百分比。若要確保結果中可以顯示所有相符的追蹤，請進一步縮小篩選條件表達式，或選擇較短的時間範圍。

若要先取得最新的結果，主控台會從時間範圍結尾開始掃描，並倒退進行。如果有大量追蹤，但僅有少數結果，主控台會將時間範圍分成區塊並平行掃描。進度列會顯示已掃描的部分時間範圍。



搭配使用篩選條件表達式與群組

群組是一系列追蹤，為篩選條件表達式所定義。您可以使用 群組來產生額外的服務圖表，並提供 Amazon CloudWatch 指標。

群組會根據名稱或 Amazon Resource Name (ARN) 進行識別，且包含篩選條件表達式。此服務會比較傳入表達式的追蹤，並依序存放。

您可以在下拉式功能表，篩選條件表達式左側的搜尋列，建立或修改群組。

> **Note**
>
> 如果服務驗證群組資格時發生錯誤，該群組即不會處理傳入的追蹤，且會記錄錯誤指標。

如需 群組的詳細資訊，請參閱 設定 群組。

篩選條件表達式語法

篩選條件表達式可以包含「關鍵字」、一元或二元「運算子」以及用於比較的「值」。

```
keyword operator value
```

不同運算子適用於不同類型的關鍵字。例如，responsetime 是數字關鍵字，可相較於與數字相關的運算子。

Example – 回應時間大於 5 秒的請求

```
responsetime > 5
```

您可以使用 AND 或 OR 運算子，將多個表達式結合成一個複合表達式。

Example – 總持續時間為 5–8 秒的請求

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

簡單關鍵字和運算子只能發現追蹤層級的問題。如果下游發生錯誤，但已由您的應用程式處理而未傳回給使用者，搜尋 error 時就不會找到此錯誤。

若要尋找含有下游問題的追蹤，您可以使用複雜的關鍵字 service() 和 edge()。這些關鍵字可讓您將篩選條件表達式套用到所有下游節點、單一下游節點，或是兩個節點之間的邊緣。如需更高的精細程度，您可以依據 id() 函數類型來篩選服務和邊緣。

布林值關鍵字

布林值關鍵字值可為 true 或 false。使用這些關鍵字來尋找導致錯誤的追蹤。

布林值關鍵字

- ok – 回應狀態碼為 2XX 成功。
- error – 回應狀態碼為 4XX 用戶端錯誤。
- throttle – 回應狀態碼為 429 太多請求。
- fault – 回應狀態碼為 5XX 伺服器錯誤。
- partial – 請求的區段不完整。
- inferred – 請求已推斷區段。
- first – Element 是列舉清單的第一個。
- last – Element 是列舉清單的最後一個。
- remote – 根本原因實體為遠端。
- root – 服務是追蹤的進入點或根區段。

布林值運算子可尋找指定索引鍵為 `true` 或 `false` 的區段。

布林值運算子

- none – 如果關鍵字為 true，則表達式為 true。
- ！– 如果關鍵字為 false，表示表達式為 true。
- =，!= – 將關鍵字的值與字串`true`或 進行比較`false`。這些運算子的作用與其他運算子相同，但更加明確。

Example – 回應狀態為 2XX OK

```
ok
```

Example – 回應狀態不是 2XX OK

```
!ok
```

Example – 回應狀態不是 2XX OK

```
ok = false
```

Example – 最後一個列舉的錯誤追蹤具有錯誤名稱 "deserialize"

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

Example – 具有遠端區段的請求，涵蓋範圍大於 0.7 且服務名稱為「行程」

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

Example – 具有推斷客群的請求，其中服務類型為「AWS：DynamoDB」

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

Example – 具有名稱為「資料平面」作為根的客群的請求

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

數字關鍵字

使用數字關鍵字以搜尋含特定回應時間、持續時間或回應狀態的請求。

數字關鍵字

- `responsetime` – 伺服器傳送回應所花費的時間。
- `duration` – 請求總持續時間，包括所有下游呼叫。
- `http.status` – 回應狀態碼。
- `index` – 列舉清單中 元素的位置。
- `coverage` – 實體回應時間相對於根區段回應時間的十進位百分比。僅適用於回應時間根本原因實體。

數字運算子

數字關鍵字使用標準的對等和比較運算子。

- =，!= – 關鍵字等於或不等於數值。
- <、<=>、 >=- 關鍵字小於或等於數值。

Example – 回應狀態不是 200 OK

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

Example – 總持續時間為 5–8 秒的請求

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

Example – 在 3 秒內成功完成的請求，包括所有下游呼叫

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

Example – 索引大於 5 的列舉清單實體

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

Example – 涵蓋範圍大於 0.8 的最後一個實體的請求

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

字串關鍵字

使用字串關鍵字以尋找請求標頭中含特定文字或特定使用者 ID 的追蹤。

字串關鍵字

- `http.url` – 請求 URL。
- `http.method` – 請求方法。
- `http.useragent` – 請求使用者代理程式字串。
- `http.clientip` – 申請者的 IP 地址。
- `user` – 追蹤中任何區段的使用者欄位值。
- `name` – 服務或例外狀況的名稱。
- `type` – 服務類型。
- `message` – 例外狀況訊息。
- `availabilityzone` – 追蹤中任何區段的 availabilityzone 欄位值。
- `instance.id` – 追蹤中任何區段的執行個體 ID 欄位值。
- `resource.arn` – 追蹤中任何區段的資源 ARN 欄位值。

字串運算子可尋找等於或包含特定文字的值。您必須一律在引號中指定值。

字串運算子

- `=`，`!=` – 關鍵字等於或不等於數值。
- `CONTAINS` – 關鍵字包含特定字串。
- `BEGINSWITH` ， `ENDSWITH` – 關鍵字以特定字串開頭或結尾。

Example – http.url 篩選條件

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

若要測試追蹤上是否存在欄位 (無論其值為何)，請檢查欄位是否包含空白字串。

Example – 使用者篩選條件

尋找所有含使用者 ID 的追蹤。

```
user CONTAINS ""
```

Example – 選取具有故障根本原因的追蹤，其中包含名為 "Auth" 的服務

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

Example – 選取具有回應時間根本原因的追蹤，其上次服務具有 DynamoDB 類型

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

Example – 選取具有故障根本原因的追蹤，其上次例外狀況具有「account_id 存取遭拒：1234567890」訊息

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

複雜關鍵字

使用複雜關鍵字以根據服務名稱、邊緣名稱或註釋值來尋找請求。若是服務和邊緣，您可以指定額外篩選條件表達式以套用到服務或邊緣。若是註釋，您可以使用布林值、數字或字串運算子，篩選含特定索引鍵的註釋值。

複雜關鍵字

- `annotation[`*`key`*`]` – 具有欄位*###*的註釋值。註釋值可以是布林值、數字或字串，所以您可以使用任何這些類型的比較運算子。您可以將此關鍵字與 `service`或 `edge`關鍵字結合使用。包含點（句點）的註釋索引鍵必須包裝在方括號（【】）中。
- `edge(`*`source`*`, `*`destination`*`) {`*`filter`*`}` – 服務*##*和*###*之間的連線。選用的大括號可包含篩選表達式，以套用到此連線的服務。
- `group.`*`name`* ` / group.`*`arn`* – 群組篩選條件表達式的值，由群組名稱或群組 ARN 參考。
- `json` – JSON 根本原因物件。如需以程式設計方式建立 JSON 實體的步驟，請參閱從 AWS X-Ray 取得資料。
- `service(`*`name`*`) {`*`filter`*`}` – 具有*##*的服務。選用的大括號可包含篩選表達式，以套用到服務所建立的區段。

使用 服務關鍵字來尋找針對追蹤映射上特定節點的請求的追蹤。

複雜關鍵字運算子會尋找已設定或未設定指定金鑰的區段。

複雜關鍵字運算子

- none – 如果已設定關鍵字，則表達式為 true。如果關鍵字是布林值類型，則會評估為布林值。
- ！– 如果未設定關鍵字，表示表達式為 true。如果關鍵字是布林值類型，則會評估為布林值。
- =，!= – 比較關鍵字的值。
- edge(*source*, *destination*) {*filter*} – 服務##和###之間的連線。選用的大括號可包含篩選表達式，以套用到此連線的服務。
- annotation[*key*] – 具有欄位###的註釋值。註釋值可以是布林值、數字或字串，所以您可以使用任何這些類型的比較運算子。您可以將此關鍵字與 service或 edge關鍵字結合使用。
- json – JSON 根本原因物件。如需以程式設計方式建立 JSON 實體的步驟，請參閱從 AWS X-Ray 取得資料。

使用 服務關鍵字來尋找針對追蹤映射上特定節點的請求的追蹤。

Example – 服務篩選條件

包含對 api.example.com 的呼叫且發生故障 (500 系列錯誤) 的請求。

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

您可以排除服務名稱，以將篩選條件表達式套用到服務地圖中的所有節點。

Example – 服務篩選條件

在您的追蹤映射上的任何位置造成錯誤的請求。

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

邊緣關鍵字可將篩選條件表達式套用到兩個節點之間的連線。

Example – 邊緣篩選條件

api.example.com 服務對 backend.example.com 發出呼叫但因錯誤而失敗的請求。

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

您也可以搭配使用 ！運算子與服務和邊緣關鍵字，以排除其他篩選條件表達式結果的服務或邊緣。

Example – 服務和請求篩選條件

URL 開頭為 `http://api.example.com/` 並包含 `/v2/` 但未到達名稱為 `api.example.com` 之服務的請求。

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

Example – 服務和回應時間篩選條件

尋找`http url`已設定 且回應時間大於 2 秒的追蹤。

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

對於註釋，您可以呼叫`annotation[`*`key`*`]`設定 的所有追蹤，或使用對應於 值類型的比較運算子。

Example – 標註字串值

含名稱為 `gameid`、字串值為 `"817DL6V0"` 之註釋的請求。

```
annotation[gameid] = "817DL6V0"
```

Example – 已設定註釋

具有名為 `age` 集合的註釋的請求。

```
annotation[age]
```

Example – 未設定註釋

沒有標註`age`集的請求。

```
!annotation[age]
```

Example – 標註數字值

含註釋存留期且數值大於 29 的請求。

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

Example – 註釋結合服務或邊緣

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

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

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

Example – 具有使用者的 群組

追蹤符合high_response_time群組篩選條件 （ 例如 responseTime > 3) 且使用者名為 Alice 的請求。

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

Example – 具有根本原因實體的 JSON

有相符根本原因實體的請求。

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

id 函數

當您將服務名稱提供給 service 或 edge 關鍵字時，您可取得具有該名稱之所有節點的結果。如需更精確的篩選，除了名稱之外，您還可以使用 id 函數來指定服務類型，以區分名稱相同的節點。

在監控帳戶中檢視來自多個帳戶的追蹤時，請使用 account.id函數指定服務的特定帳戶。

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

您可以使用 id 函數來代替服務和邊緣篩選條件中的服務名稱。

```
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 函數會在追蹤映射中產生兩個節點；一個用於函數叫用，另一個用於 Lambda 服務。這兩個節點的名稱相同，但類型不同。標準的服務篩選條件可尋找這兩種追蹤。

Example – 服務篩選條件

包含任何名為 random-name 服務上的錯誤的請求。

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

使用 id 函數來縮小搜尋範圍至函數本身的錯誤，而不含服務的錯誤。

Example – 具有 ID 函數的服務篩選條件

包含名為 random-name、類型為 AWS::Lambda::Function 服務上的錯誤的請求。

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

若要依據類型來搜尋節點，您也可以完全排除名稱。

Example – 具有 ID 函數和服務類型的服務篩選條件

包含類型為 AWS::Lambda::Function 服務上的錯誤的請求。

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

若要搜尋特定節點 AWS 帳戶，請指定帳戶 ID。

Example – 具有 ID 函數和帳戶 ID 的服務篩選條件

在特定帳戶 ID 內包含服務的請求AWS::Lambda::Function。

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

跨帳戶追蹤

AWS X-Ray 支援跨帳戶可觀測性，可讓您監控和故障診斷 中跨多個帳戶的應用程式 AWS 區域。您可以順暢地搜尋、視覺化和分析任何連結帳戶中的指標、日誌和追蹤，就像您在單一帳戶中操作一樣。這提供跨多個帳戶周遊的請求的完整檢視。您可以在 CloudWatch 主控台的 X-Ray 追蹤地圖和追蹤頁面中檢視跨帳戶追蹤。

共用的可觀測性資料可包含下列任一類型的遙測：

- Amazon CloudWatch 中的指標
- Amazon CloudWatch Logs 中的日誌群組
- 中的追蹤 AWS X-Ray
- Amazon CloudWatch Application Insights 中的應用程式

設定跨帳戶可觀測性

若要開啟跨帳戶可觀測性，請設定一或多個 AWS 監控帳戶，並將其連結至多個來源帳戶。監控帳戶是中央帳戶 AWS 帳戶 ，可檢視來源帳戶產生的可觀測性資料並與之互動。來源帳戶是針對其中包含的資源 AWS 帳戶 產生可觀測性資料的個人。

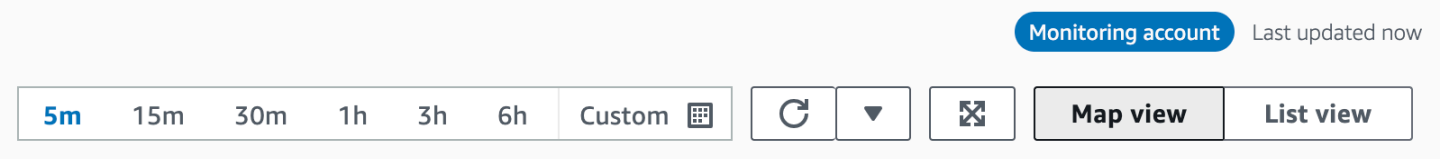
來源帳戶與監控帳戶共用其可觀測性資料。追蹤會從每個來源帳戶複製到最多五個監控帳戶。從來源帳戶到第一個監控帳戶的追蹤複本是免費的。傳送至其他監控帳戶的追蹤複本會根據標準定價，向每個來源帳戶收費。如需詳細資訊，請參閱 AWS X-Ray 定價和 Amazon CloudWatch 定價。

若要在監控帳戶和來源帳戶之間建立連結，請使用 CloudWatch 主控台或 AWS CLI 和 API 中的新可觀測性存取管理員命令。如需詳細資訊，請參閱 CloudWatch 跨帳戶觀察功能。

> **Note**
>
> X-Ray 追蹤的計費方式是收到追蹤 AWS 帳戶 的 。如果抽樣請求跨越多個 服務 AWS 帳戶 ，每個帳戶都會記錄個別的追蹤，且所有追蹤都會共用相同的追蹤 ID。若要進一步了解跨帳戶可觀測性定價，請參閱AWS X-Ray 定價和 Amazon CloudWatch 定價。

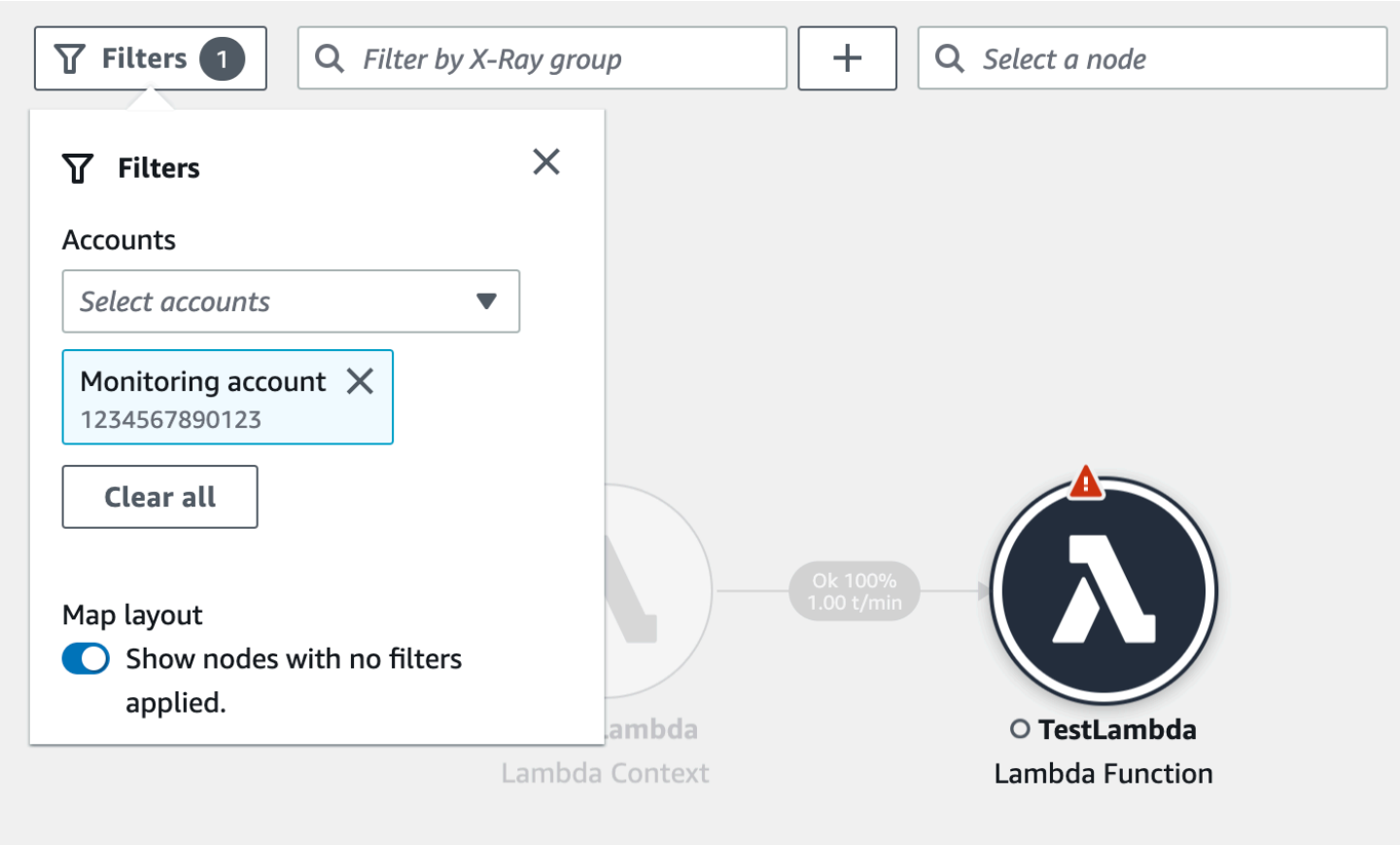
檢視跨帳戶追蹤

跨帳戶追蹤會顯示在監控帳戶中。每個來源帳戶只會顯示該特定帳戶的本機追蹤。下列各節假設您已登入監控帳戶，並已開啟 Amazon CloudWatch 主控台。在追蹤地圖和追蹤頁面上，監控帳戶徽章會顯示在右上角。

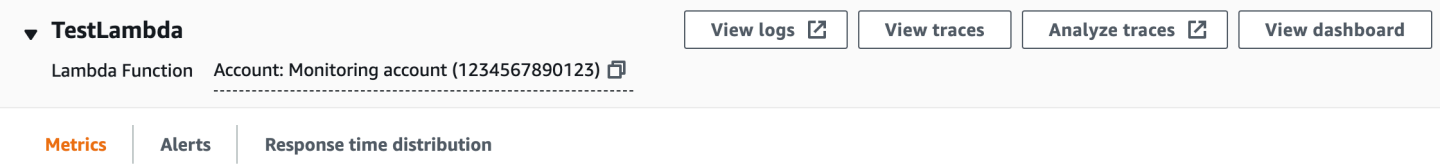


追蹤映射

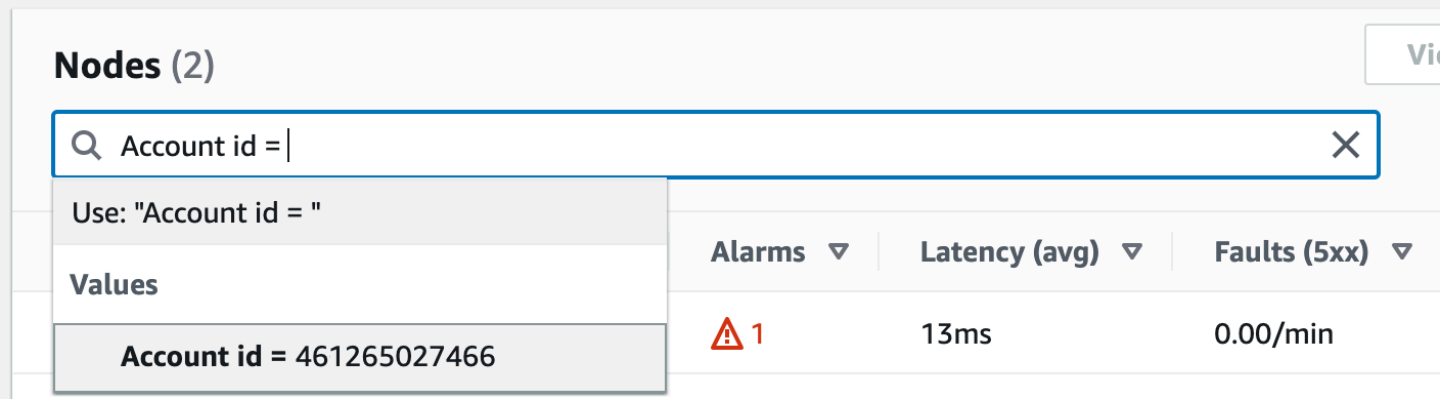
在 CloudWatch 主控台中，從左側導覽窗格選擇 X-Ray 追蹤下的追蹤映射。根據預設，追蹤映射會顯示所有來源帳戶的節點，這些來源帳戶會將追蹤傳送至監控帳戶，以及監控帳戶本身的節點。在追蹤映射上，選擇左上角的篩選條件，使用帳戶下拉式清單篩選追蹤映射。套用帳戶篩選條件後，來自不符合目前篩選條件之帳戶的 服務節點會顯示為灰色。



當您選擇服務節點時，節點詳細資訊窗格會包含服務的帳戶 ID 和標籤。



在追蹤映射的右上角，選擇清單檢視以查看服務節點清單。服務節點清單包含來自監控帳戶和所有已設定來源帳戶的 服務。從節點篩選條件中選擇節點，依帳戶標籤或帳戶 ID 篩選節點清單。

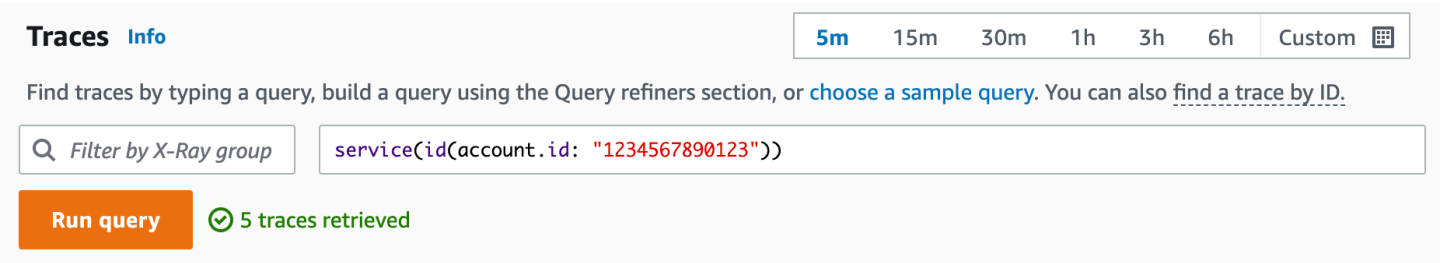


追蹤

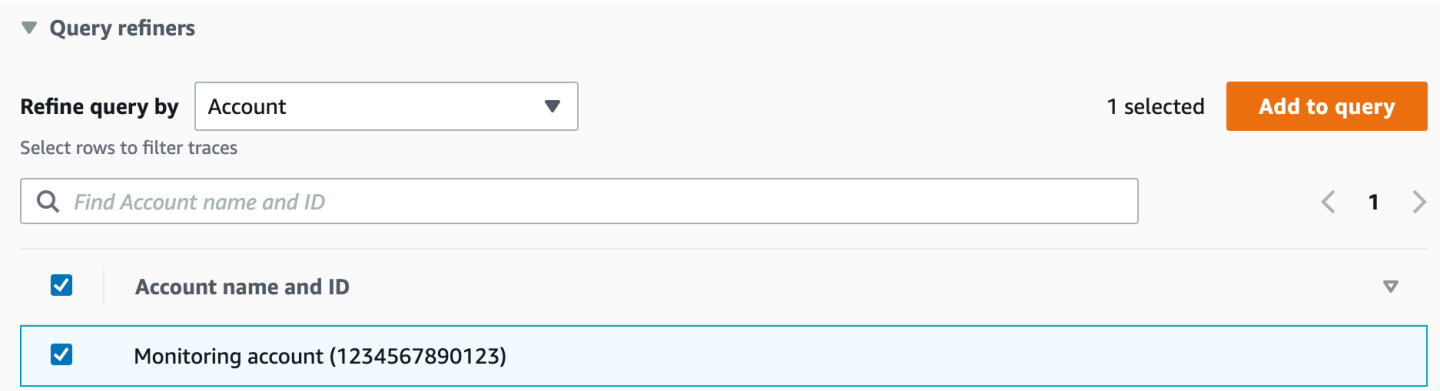
從監控帳戶開啟 CloudWatch 主控台，然後在左側導覽窗格中的 X-Ray 追蹤下選擇追蹤，以檢視跨多個帳戶的追蹤詳細資訊。 您也可以在 X-Ray 追蹤地圖中選擇節點，然後從節點詳細資訊窗格中選擇檢視追蹤，以開啟此頁面。

追蹤頁面支援依帳戶 ID 查詢。若要開始使用，請輸入包含一或多個帳戶 ID 的查詢。 IDs 下列範例查詢已透過帳戶 ID X 或 Y 傳遞的追蹤：

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



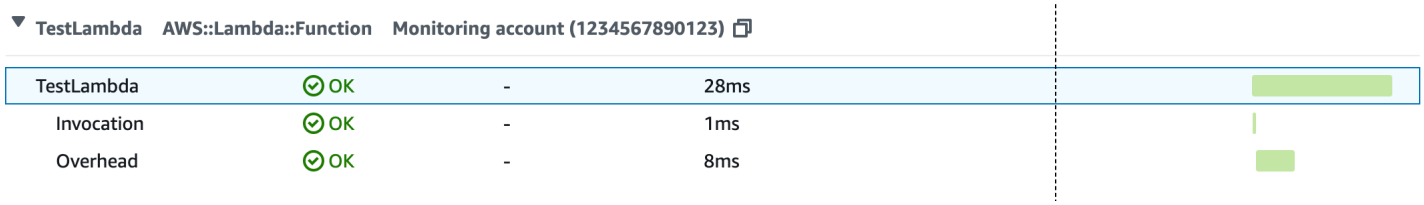
依帳戶精簡查詢。從清單中選擇一或多個帳戶，然後選擇新增至查詢。



追蹤詳細資訊

從追蹤頁面底部的追蹤清單中選擇追蹤，以檢視追蹤的詳細資訊。追蹤詳細資訊隨即顯示，包括追蹤詳細資訊映射來自所有帳戶的服務節點，而這些帳戶已傳遞追蹤。選擇特定的服務節點以查看其對應的帳戶。

客群時間軸區段會顯示時間軸中每個客群的帳戶詳細資訊。



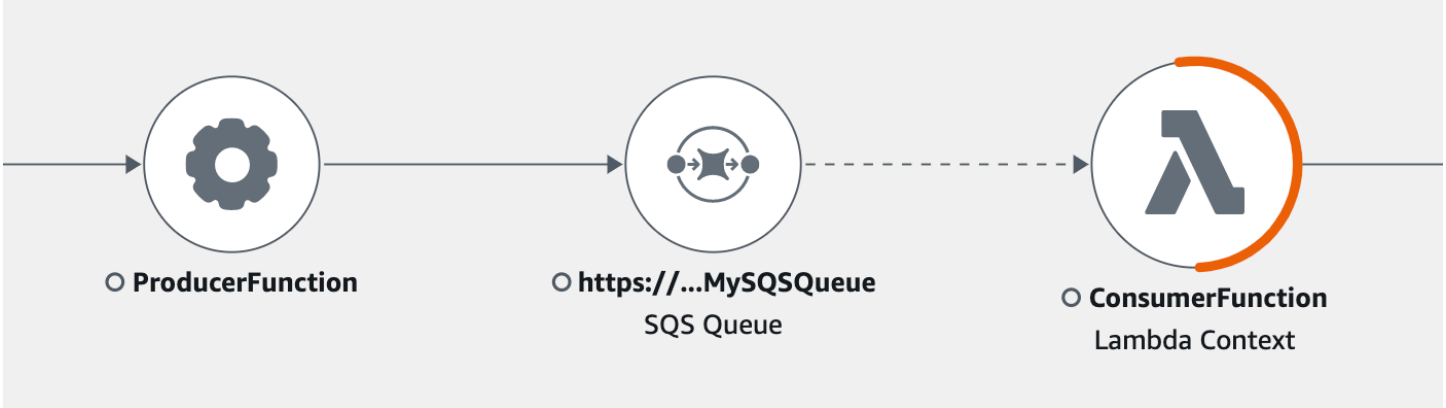
追蹤事件驅動型應用程式

AWS X-Ray 支援使用 Amazon SQS 和 追蹤事件驅動型應用程式 AWS Lambda。使用 CloudWatch 主控台查看與 Amazon SQS 佇列且由一或多個 Lambda 函數處理的每個請求的連線檢視。來自上游訊息生產者的追蹤會自動連結至來自下游 Lambda 消費者節點的追蹤，建立end-to-end檢視。

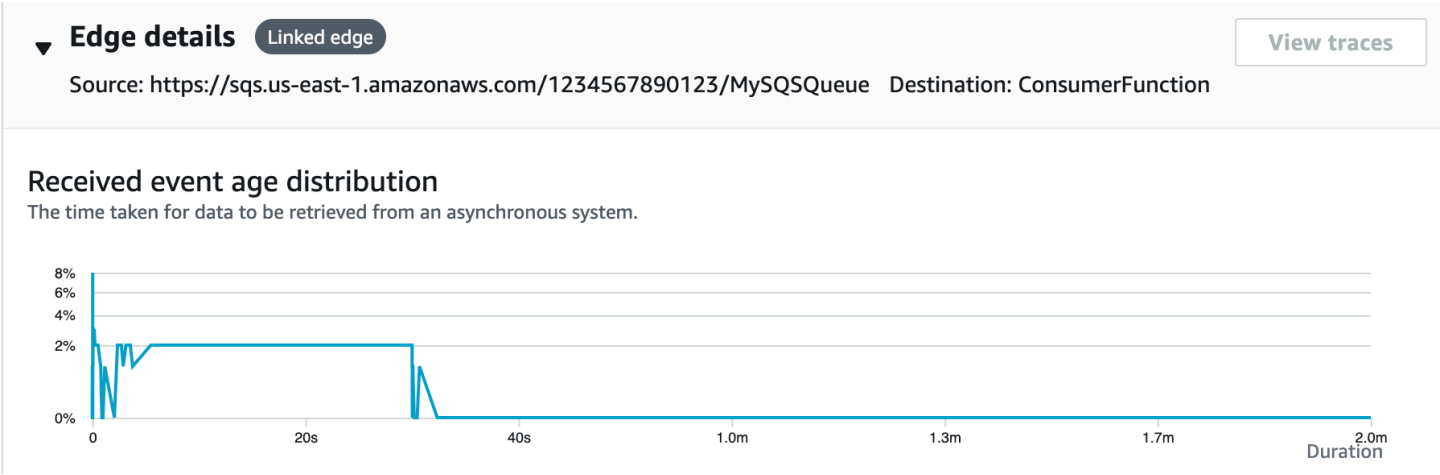
> **Note**
>
> 每個追蹤區段最多可連結至 20 個追蹤，而追蹤最多可包含 100 個連結。在某些情況下，連結其他追蹤可能會導致超過追蹤文件大小上限，進而導致追蹤可能不完整。例如，當啟用追蹤的 Lambda 函數在單一調用中將許多 SQS 訊息傳送至佇列時，可能會發生這種情況。如果您遇到此問題，可以使用 X-Ray SDKs進行緩解。如需詳細資訊，請參閱適用於 Java、Node.js、Python、Go 或 .NET 的 X-Ray 開發套件。

在追蹤映射中檢視連結的追蹤

使用 CloudWatch 主控台中的追蹤映射頁面，檢視追蹤映射，其中包含來自訊息生產者且連結至 Lambda 消費者追蹤的追蹤。這些連結會以虛線邊緣顯示，以連接 Amazon SQS 節點和下游 Lambda 取用者節點。



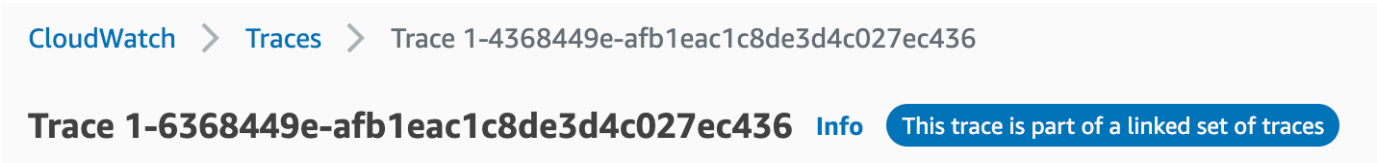
選取虛線邊緣以顯示收到的事件存留期長條圖，其會映射消費者收到事件存留期的分散。每次收到事件時都會計算存留期。



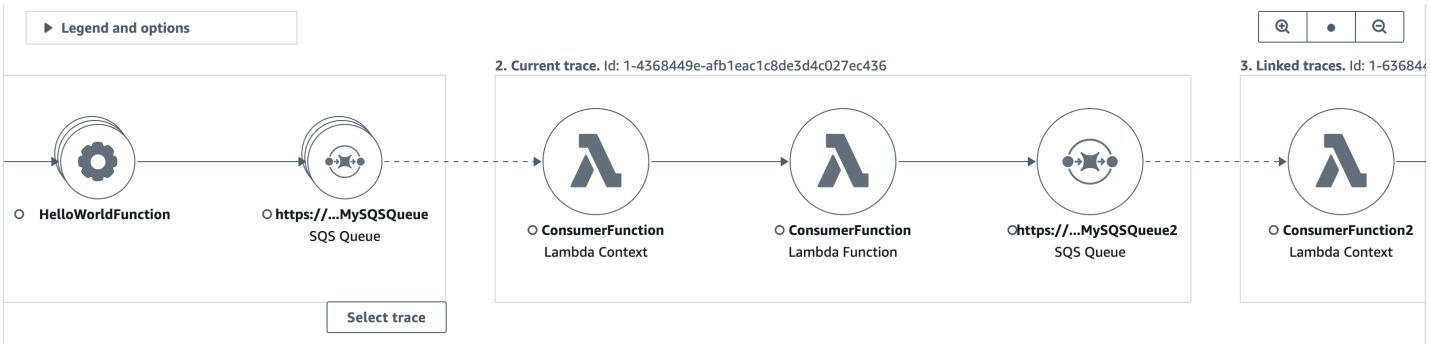
檢視連結的追蹤詳細資訊

檢視從訊息生產者、Amazon SQS 佇列或 Lambda 取用者傳送的追蹤詳細資訊：

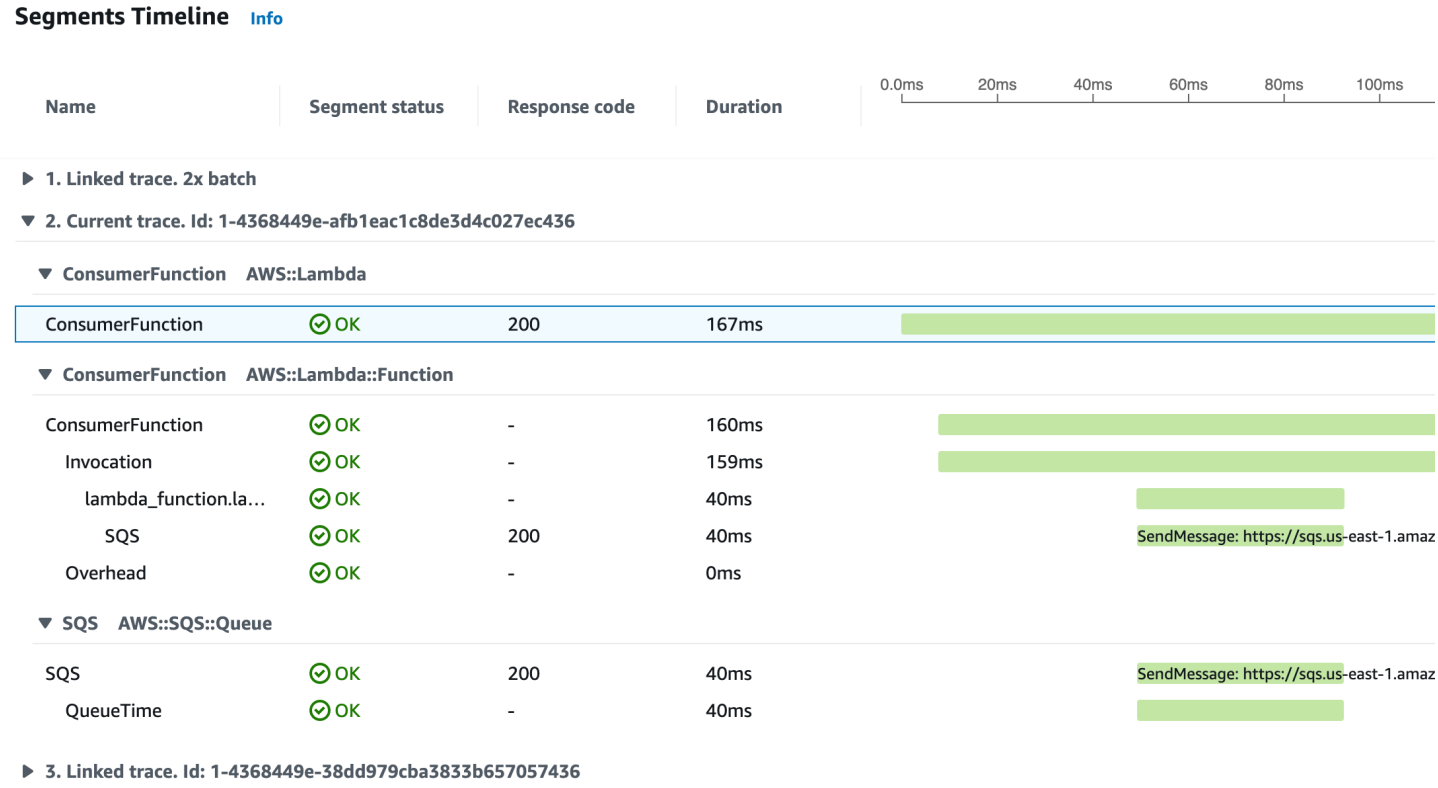
1. 使用追蹤地圖來選取訊息生產者、Amazon SQS 或 Lambda 取用者節點。
2. 從節點詳細資訊窗格中選擇檢視追蹤，以顯示追蹤清單。您也可以直接導覽至 CloudWatch 主控台中的追蹤頁面。
3. 從清單中選擇特定追蹤，以開啟追蹤詳細資訊頁面。當選取的追蹤是一組連結追蹤的一部分時，追蹤詳細資訊頁面會顯示訊息。



追蹤詳細資訊映射會顯示目前的追蹤，以及上游和下游連結追蹤，每個追蹤都包含在指示每個追蹤邊界的方塊中。如果目前選取的追蹤連結至多個上游或下游追蹤，則會堆疊上游或下游連結追蹤內的節點，並顯示選取追蹤按鈕。



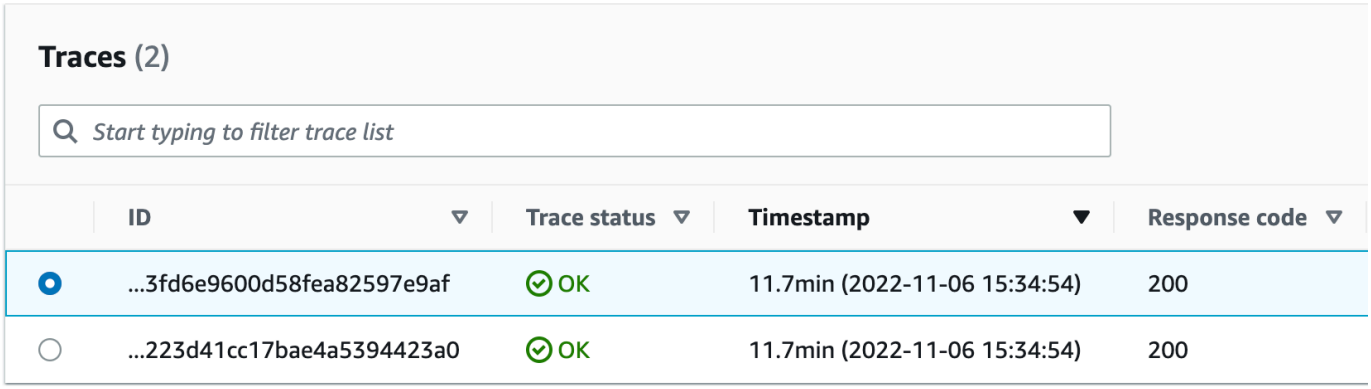
在追蹤詳細資訊映射下方，會顯示追蹤區段的時間軸，包括上游和下游連結追蹤。如果有多個上游或下游連結追蹤，則無法顯示其區段詳細資訊。若要檢視一組連結追蹤中單一追蹤的區段詳細資訊，請選取單一追蹤，如下所述。



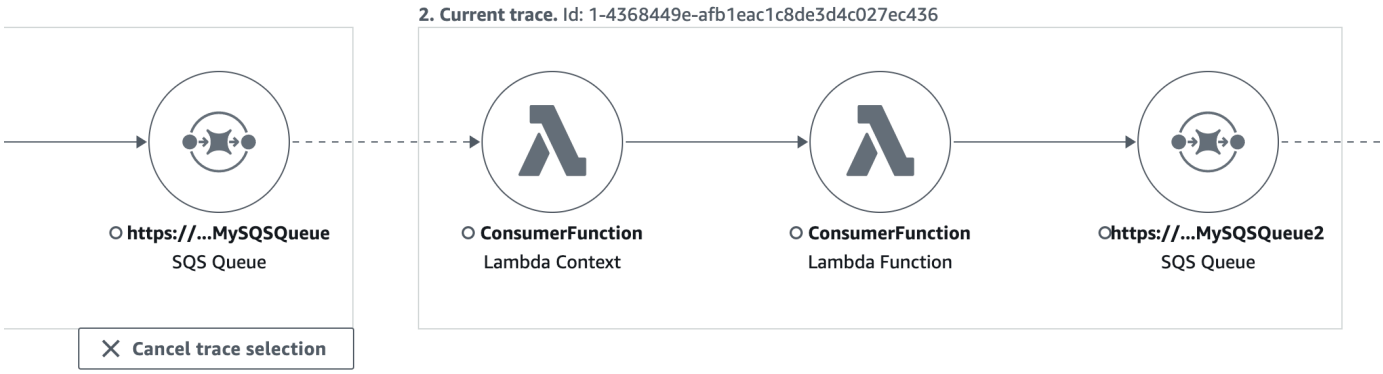
在一組連結追蹤中選取單一追蹤

將一組連結的追蹤篩選為單一追蹤，以查看時間軸中的區段詳細資訊。

1. 選擇追蹤詳細資訊地圖上連結追蹤下方的選取追蹤。追蹤清單隨即顯示。



2. 選取追蹤旁的選項按鈕，以在追蹤詳細資訊映射中檢視。
3. 選擇取消追蹤選擇，以檢視整組連結的追蹤。



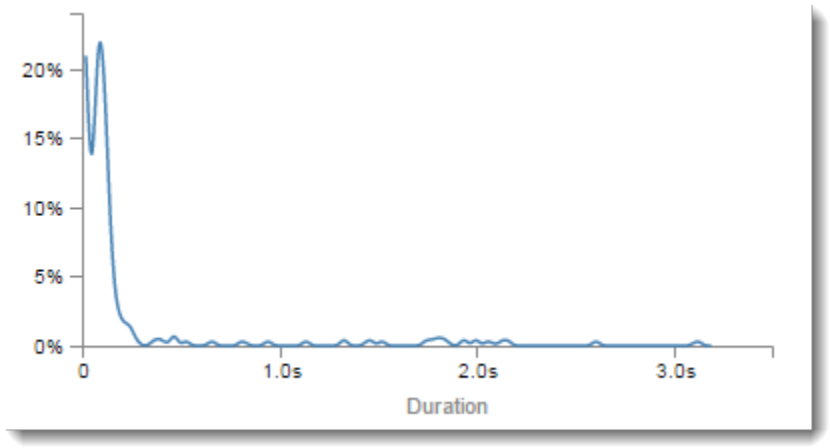
使用延遲長條圖

當您在 AWS X-Ray 追蹤映射上選取節點或邊緣時，X-Ray 主控台會顯示延遲分佈長條圖。

Latency (延遲)

延遲是指請求開始和完成之間的時間。長條圖中顯示了延遲分佈。它會在 x 軸上顯示持續時間，在 y 軸上顯示符合每個持續時間的請求百分比。

此長條圖顯示的服務可在 300 毫秒內完成大多數請求。少許百分比的請求花費最多 2 秒，以及幾個異常值花費更多時間。



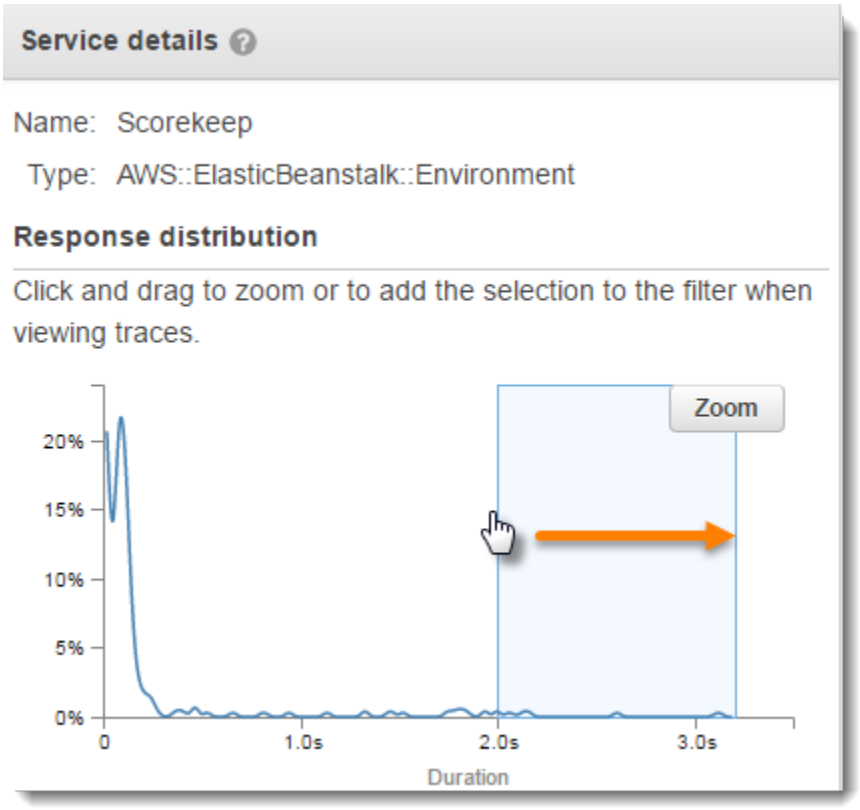
解譯服務詳細資訊

服務長條圖和邊緣長條圖可從服務或申請者的角度，提供延遲的視覺化呈現。

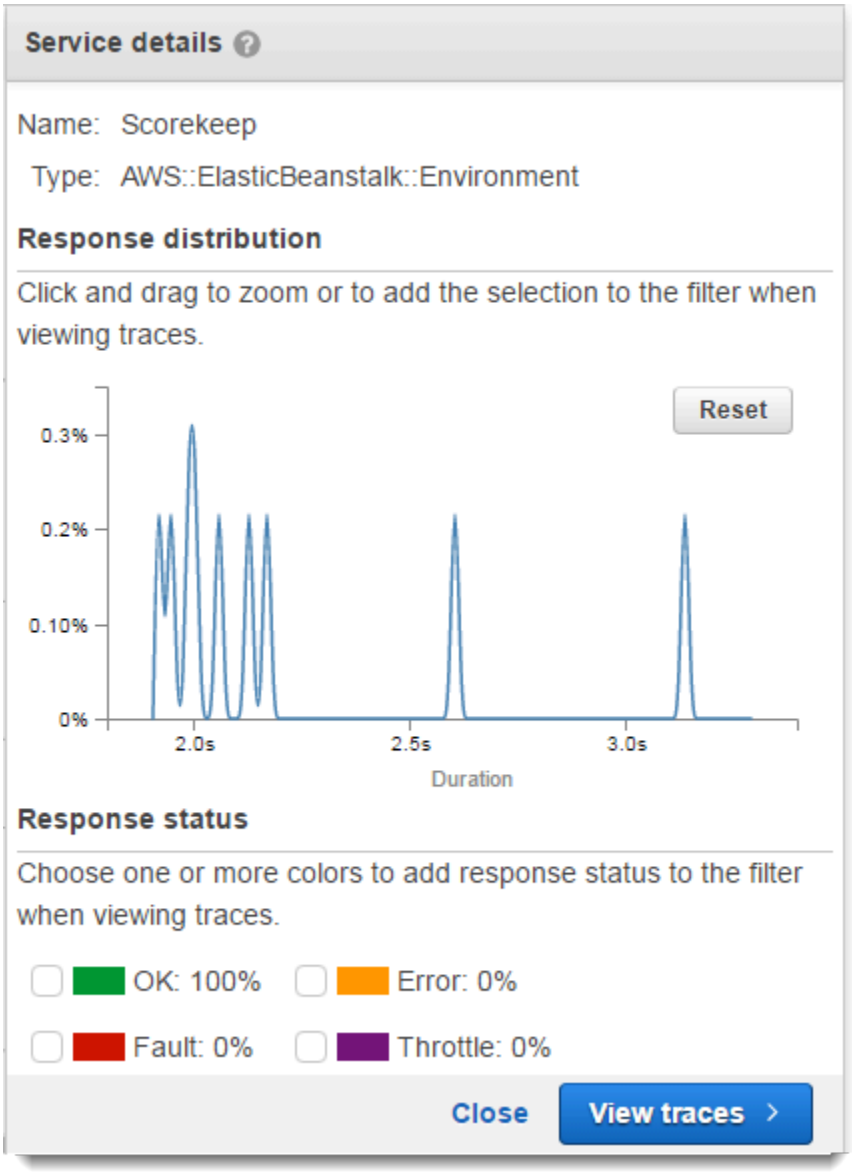
- 按一下圓圈以選擇服務節點。X-Ray 會顯示服務所服務請求的長條圖。延遲是由服務記錄的延遲，且不包含任何服務和申請者之間的網路延遲。
- 按一下兩個服務之間的邊緣線或箭頭尖端，以選擇邊緣。X-Ray 會顯示由下游服務所提供服務之請求者的請求長條圖。這些延遲是由申請者記錄的延遲，且包含兩個服務之間的網路連線延遲。

若要解譯 Service details (服務詳細資訊) 長條圖，您可以尋找與長條圖中大多數值差異最大的值。您可將這些「異常值」視為長條圖中的峰值，並檢視特定區域的追蹤資料以調查發生的情況。

若要查看依延遲篩選的追蹤，請在長條圖上選取範圍。按一下您想要選擇的起點位置，並從左到右拖曳，將要包含在追蹤篩選條件中的延遲範圍反白顯示。



選取範圍之後，您可以選擇 Zoom (縮放)，以僅檢視部分長條圖並縮小您的選擇範圍。



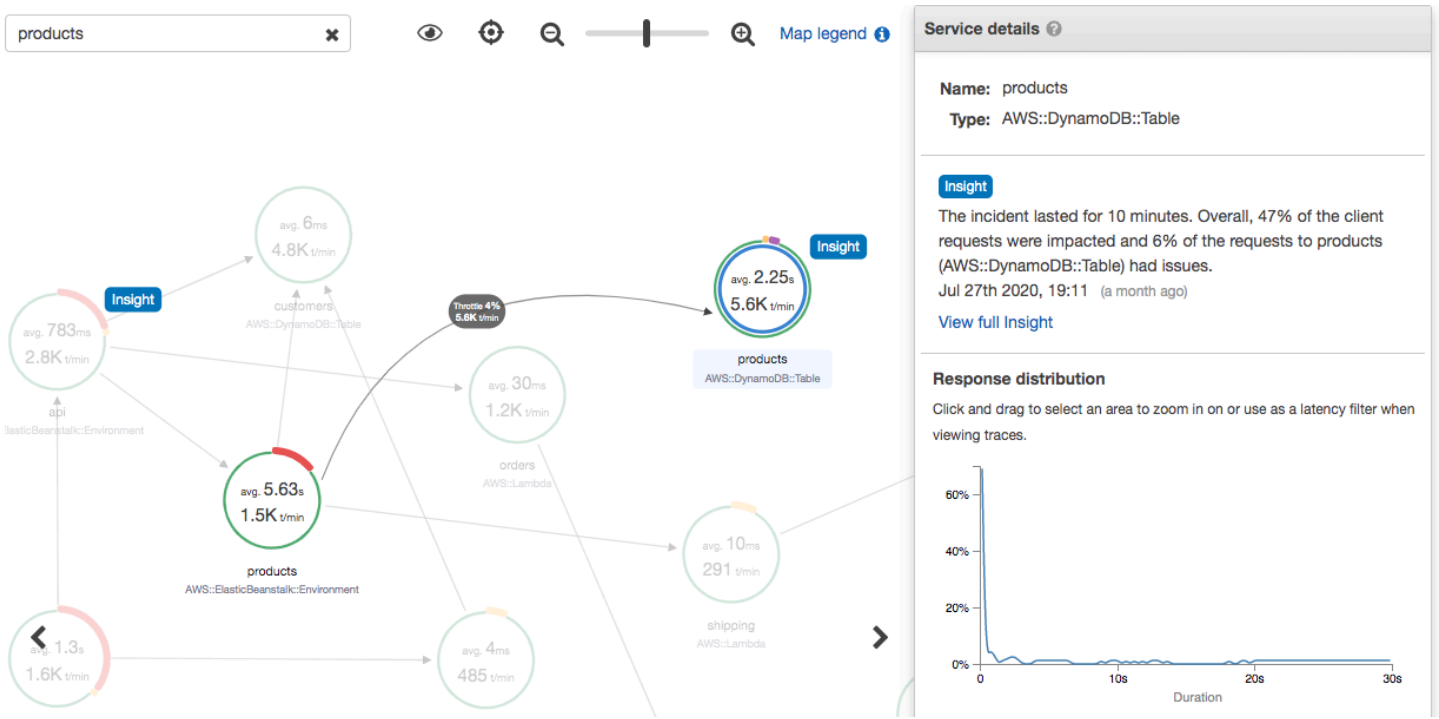
一旦您設定好所需的焦點區域時，請選擇 View traces (檢視追蹤)。

使用 X-Ray 洞察

AWS X-Ray 會持續分析您帳戶中的追蹤資料，以識別應用程式中出現的問題。當錯誤率超過預期範圍時，它會建立洞見，記錄問題並追蹤其影響，直到問題解決為止。透過洞見，您可以：

- 識別應用程式問題發生的位置、問題的根本原因，以及相關的影響。洞見提供的影響分析可讓您衍生問題的嚴重性和優先順序。
- 當問題隨時間變更時接收通知。Insights 通知可以使用 Amazon EventBridge 與您的監控和提醒解決方案整合。此整合可讓您根據問題的嚴重性來傳送自動電子郵件或提醒。

X-Ray 主控台會在追蹤映射中識別具有持續事件的節點。若要查看洞見的摘要，請選擇受影響的節點。您也可以從左側的導覽窗格中選擇 Insights，以檢視和篩選洞見。



當 X-Ray 在服務的一或多個節點中偵測到異常時，會建立洞見。服務使用統計建模來預測應用程式中服務的預期錯誤率。在上述範例中，異常是故障的來源增加 AWS Elastic Beanstalk。Elastic Beanstalk 伺服器發生多個 API 呼叫逾時，導致下游節點異常。

在 X-Ray 主控台中啟用洞見

必須針對您要使用洞見功能的每個群組啟用洞見。您可以從群組頁面啟用洞見。

1. 開啟 X-Ray 主控台。
2. 選擇建立群組以選取現有群組或建立新的群組，然後選取啟用洞見。如需在 X-Ray 主控台中設定群組的詳細資訊，請參閱 設定 群組。
3. 在左側導覽窗格中，選擇 Insights，然後選擇要檢視的洞見。

> **Note**
>
> X-Ray 使用 GetInsightSummaries、GetInsight、GetInsightEvents 和 GetInsightImpactGraph API 操作從洞見擷取資料。若要檢視洞見，請使用 AWSXrayReadOnlyAccess IAM 受管政策，或將下列自訂政策新增至您的 IAM 角色：
>
> ```
> {
>    "Version": "2012-10-17",
>    "Statement": [
>        {
>            "Effect": "Allow",
>            "Action": [
>                "xray:GetInsightSummaries",
>                "xray:GetInsight",
>                "xray:GetInsightEvents",
>                "xray:GetInsightImpactGraph"
>            ],
>            "Resource": [
>                "*"
>            ]
>        }
>    ]
> }
> ```
>
> 如需詳細資訊，請參閱AWS X-Ray 如何使用 IAM。

啟用洞見通知

透過洞見通知，會為每個洞見事件建立通知，例如建立洞見、大幅變更或關閉。客戶可以透過 Amazon EventBridge 事件接收這些通知，並使用條件式規則採取動作，例如 SNS 通知、Lambda 調用、將訊息發佈至 SQS 佇列，或任何 EventBridge 支援的目標。Insights 通知會盡最大努力發出，但無法保證。如需目標的詳細資訊，請參閱 Amazon EventBridge 目標。

您可以從群組頁面啟用任何已啟用洞察的群組的洞察通知。

啟用 X-Ray 群組的通知

1. 開啟 X-Ray 主控台。

2. 選擇建立群組來選取現有群組或建立新的群組，確保已選取啟用洞見，然後選取啟用通知。如需在 X-Ray 主控台中設定群組的詳細資訊，請參閱 設定 群組。

設定 Amazon EventBridge 條件式規則

1. 開啟 Amazon EventBridge 主控台。
2. 導覽至左側導覽列中的規則，然後選擇建立規則。
3. 提供規則的名稱和描述。
4. 選擇事件模式，然後選擇自訂模式。提供包含 `"source": [ "aws.xray" ]`和 的模式`"detail-type": [ "AWS X-Ray Insight Update" ]`。以下是一些可能模式的範例。
   - 符合 X-Ray 洞察的所有傳入事件的事件模式：

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

   - 符合指定 **`state`**和 的事件模式**`category`**：

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

5. 選取並設定當事件符合此規則時，您要叫用的目標。
6. （選用）提供標籤以更輕鬆地識別和選取此規則。
7. 選擇 Create (建立)。

> **Note**
>
> X-Ray 洞察通知會將事件傳送至 Amazon EventBridge，目前不支援客戶受管金鑰。如需詳細資訊，請參閱中的資料保護 AWS X-Ray。

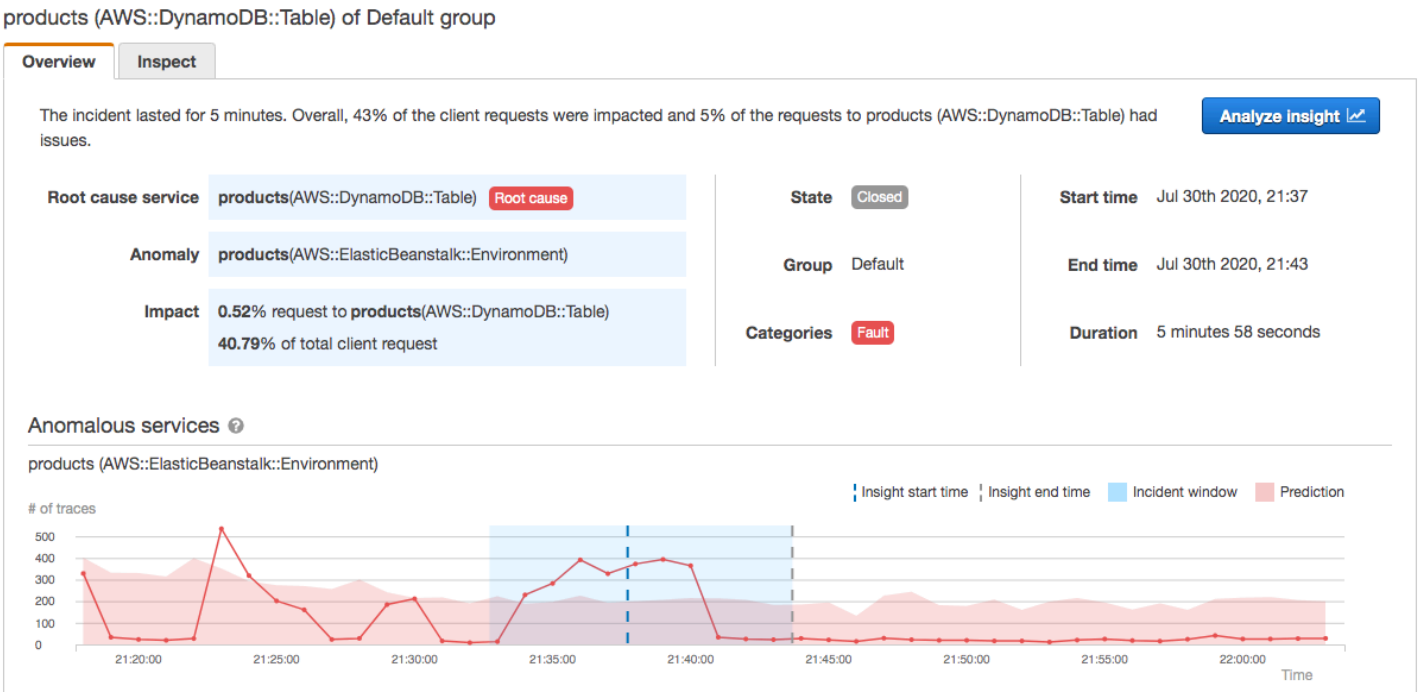
Insight 概觀

洞察嘗試回答三個關鍵問題的概觀頁面：

- 基礎問題是什麼？
- 根本原因是什麼？
- 有什麼影響？

異常服務區段顯示每個服務的時間軸，說明事件期間的錯誤率變更。時間軸會顯示覆蓋在實心頻帶上的故障追蹤數目，根據記錄的流量來指出預期的故障數目。洞見的持續時間由事件視窗視覺化。當 X-Ray 觀察到指標變得異常，並在洞見作用中時持續存在時，事件時段就會開始。

下列範例顯示導致事件的故障增加：

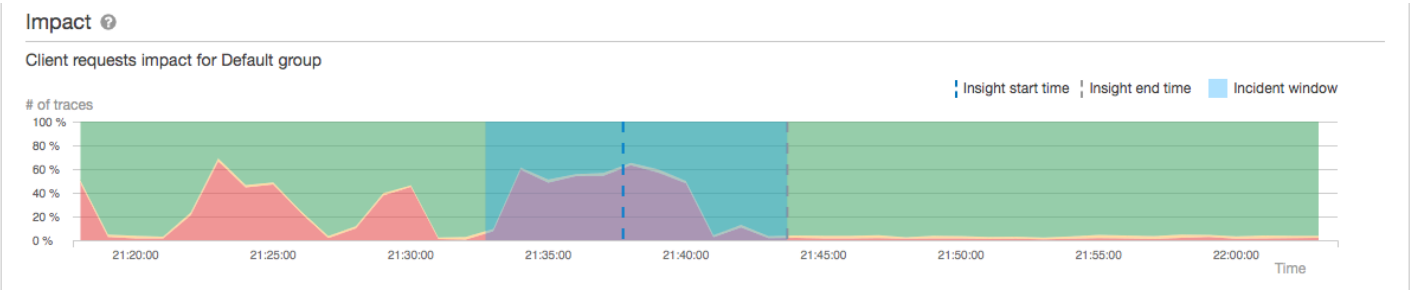


根本原因區段顯示專注於根本原因服務和受影響路徑的追蹤映射。您可以透過選取根本原因映射右上角的眼睛圖示來隱藏未受影響的節點。根本原因服務是 X-Ray 識別異常的最遠下游節點。它可以代表您

檢測的服務，或您的服務使用受檢測的用戶端呼叫的外部服務。例如，如果您使用經檢測的 AWS SDK 用戶端呼叫 Amazon DynamoDB，DynamoDB 的故障增加會導致 DynamoDB 作為根本原因的洞見。

若要進一步調查根本原因，請選取根本原因圖表上的檢視根本原因詳細資訊。您可以使用分析頁面來調查根本原因和相關訊息。如需詳細資訊，請參閱與 Analytics 主控台互動。

在地圖中繼續上游的故障可能會影響多個節點並導致多個異常。如果故障一直傳遞給發出請求的使用者，則結果為用戶端故障。這是追蹤映射根節點中的錯誤。影響圖表提供整個群組的用戶端體驗時間軸。此體驗是根據下列狀態的百分比計算：Fault、 Error、 Throttle 和 Okay。



此範例顯示在事件發生期間根節點發生故障的追蹤增加。下游服務中的事件不一定對應於用戶端錯誤增加。

選擇分析洞見會在視窗中開啟 X-Ray Analytics 主控台，您可以在其中深入探索導致洞見的追蹤集。如需詳細資訊，請參閱與 Analytics 主控台互動。

了解影響

AWS X-Ray 在產生洞見和通知時， 會測量持續問題所造成的影響。影響的測量方式有兩種：

- 對 X-Ray 群組的影響
- 對根本原因服務的影響

此影響取決於在特定期間內失敗或導致錯誤的請求百分比。此影響分析可讓您根據特定案例衍生問題的嚴重性和優先順序。除了洞察通知之外，此影響也可用於主控台體驗。

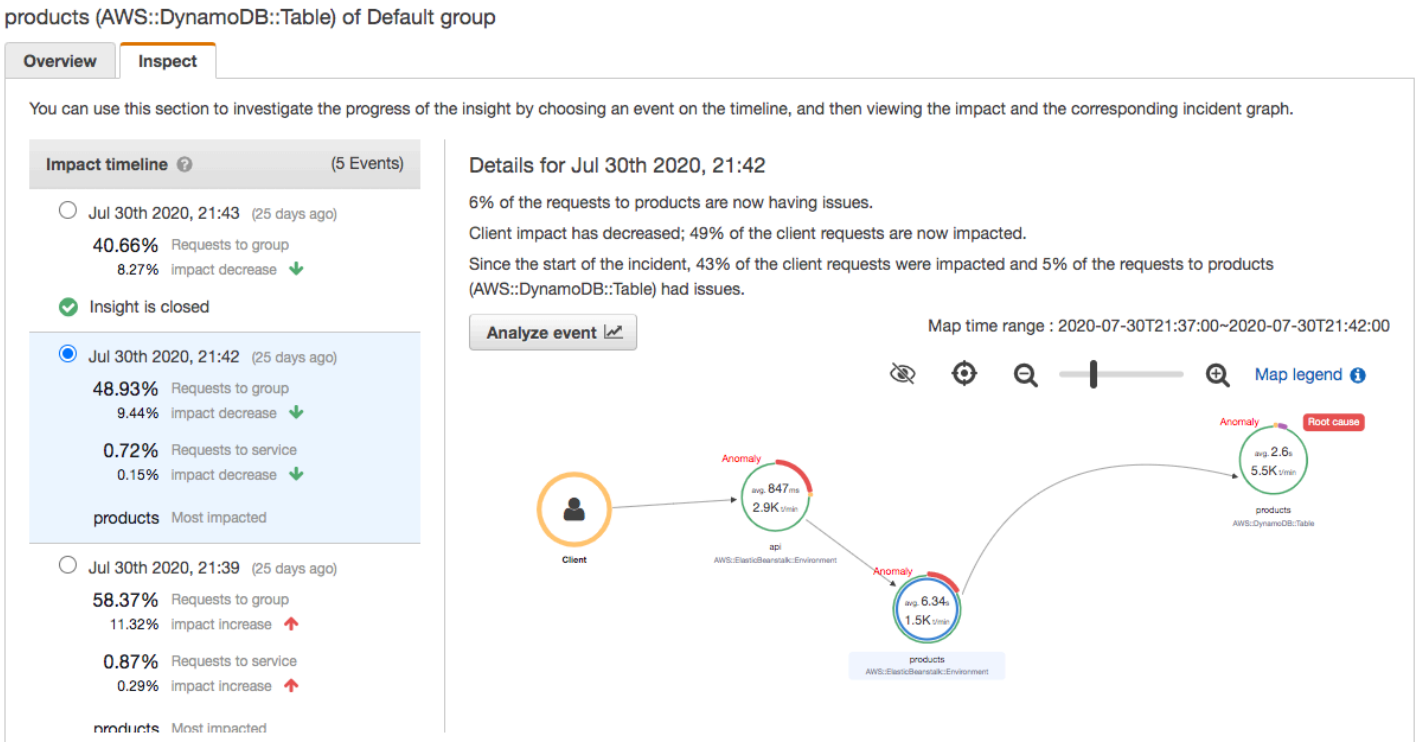
重複資料刪除

AWS X-Ray 洞見可消除多個微服務的問題。它使用異常偵測來判斷問題根本原因的服務、判斷其他相關服務是否因相同的根本原因而出現異常行為，並將結果記錄為單一洞見。

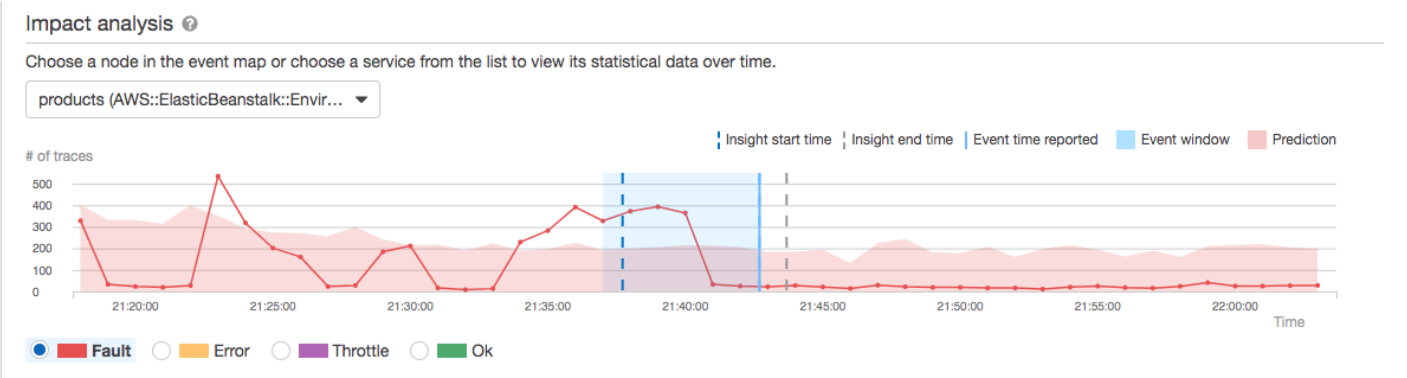
檢閱洞見的進度

X-Ray 會定期重新評估洞見，直到解決為止，並將每個值得注意的中繼變更記錄為通知，以 Amazon EventBridge 事件的形式傳送。這可讓您建置程序和工作流程，以判斷問題如何隨著時間而變更，並採取適當的動作，例如傳送電子郵件或使用 EventBridge 與提醒系統整合。

您可以在檢查頁面的影響時間軸中檢閱事件。根據預設，時間軸會顯示最受影響的服務，直到您選擇不同的服務為止。



若要查看事件的追蹤地圖和圖形，請從影響時間軸中選擇它。追蹤映射會顯示應用程式中受事件影響的服務。在影響分析下，圖形會顯示所選節點和群組中用戶端的故障時間表。



若要深入了解事件中涉及的追蹤，請在檢查頁面上選擇分析事件。您可以使用分析頁面來精簡追蹤清單，並識別受影響的使用者。如需詳細資訊，請參閱與 Analytics 主控台互動。

與 Analytics 主控台互動

AWS X-Ray Analytics 主控台是一種互動式工具，可用來解譯追蹤資料，以快速了解應用程式及其基礎服務的效能。主控台可讓您透過互動的回應時間和時間序列圖表，探索、分析和視覺化追蹤。

在 Analytics 主控台中選取選項時，主控台會建構篩選條件以反映所有追蹤的所選子集。您可以按一下與目前追蹤集相關聯之指標和欄位的圖表和面板，使用愈益精細的篩選條件來精簡作用中的資料集。

主題

- 主控台功能
- 回應時間分佈
- 時間序列活動
- 工作流程範例
- 觀察服務圖中的故障
- 識別回應時間高峰
- 檢視有狀態碼標記的所有追蹤
- 檢視在子群組中並與使用者相關聯的所有項目
- 比較兩個具有不同篩選標準的追蹤集
- 識別感興趣的追蹤以及檢視其詳細資訊

主控台功能

X-Ray Analytics 主控台使用下列主要功能來分組、篩選、比較和量化追蹤資料。

功能

功能	描述
Groups (群組)	最初選取的群組是 `Default`。若要變更已擷取的群組，請從主要篩選條件表達式搜尋列右側的選單中，選取不同的群組。若要進一步了解群組，請參閱搭配使用篩選條件表達式與群組。
Retrieved traces (已擷取的追蹤)	根據預設，Analytics 主控台會根據所選群組中的所有追蹤產生圖表。已擷取的追蹤代表您工作集中的所有追蹤。您可在此圖標中找到追蹤計數。您套用到主要搜尋列的篩選條件表達式會精簡並更新擷取的追蹤。
Show in charts/Hide from charts (在圖表中顯示/隱藏)	切換比較作用中的群組和已擷取的追蹤。若要針對任何作用中的篩選條件比較群組的相關資料，請選擇 Show in charts (在圖表中顯示)。若要從圖表中移除此檢視，請選擇 Hide from charts (在圖表中隱藏)。
Filtered trace set A (篩選過的追蹤集 A)	透過與圖形和資料表的互動，套用篩選條件來建立已篩選追蹤集 A 的條件。 套用篩選條件時，會從此圖磚中計算適用的追蹤數量，以及擷取的追蹤總數百分比。在 Filtered trace set A (篩選過的追蹤集 A) 圖標中，篩選條件填入為標籤，也可自圖標中移除。
Refine (精簡)	此功能會根據套用到追蹤集 A 的篩選條件，更新擷取的追蹤集。縮小擷取的追蹤集範圍會更新根據追蹤集 A 篩選條件擷取之所有追蹤的初步集合。已擷取追蹤的初步集合是群組中所有追蹤的抽樣子集。
Filtered trace set B (篩選過的追蹤集 B)	建立時，篩選的追蹤集 B 是篩選的追蹤集 A 的副本。 若要比較兩個追蹤集，請進行新的篩選條件選擇，以套用至追蹤集 B，而追蹤集 A 保持固定。套用篩選條件後，就會在此圖標內計

功能	描述
	算擷取自總量的適用追蹤數目與追蹤百分比。在 Filtered trace set B (篩選過的追蹤集 B) 圖標中，篩選條件填入為標籤，也可自圖標中移除。
Response time root cause entity paths (回應時間根本原因實體路徑)	記錄實體路徑的資料表。X-Ray 會判斷追蹤中哪個路徑最有可能導致回應時間。格式指出實體遇到的階層，以回應時間根本原因結束。使用這些資料列篩選重複出現的回應時間故障。如需自訂根本原因篩選條件以及透過 API 取得資料的詳細資訊，請參閱擷取和縮小範圍根本原因分析。
Delta (�)	當追蹤集 A 和追蹤集 B 都為作用中時，新增到指標表的欄。Delta 欄計算追蹤集 A 和追蹤集 B 之間的追蹤百分比差異。

回應時間分佈

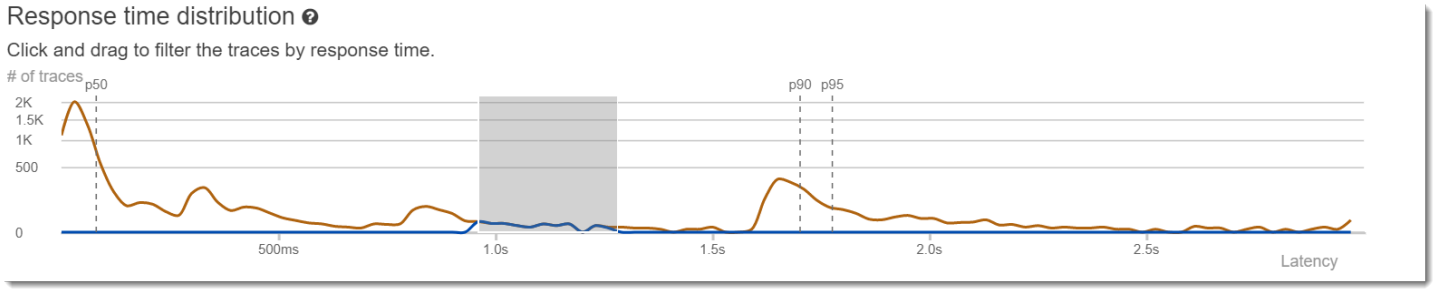
X-Ray Analytics 主控台會產生兩個主要圖形，協助您視覺化追蹤：回應時間分佈和時間序列活動。本節和下節各提供一個範例，說明閱讀圖表的基本知識。

以下是與回應時間折線圖相關聯的顏色 (時間序列圖使用相同的顏色組合)：

- 群組中的所有追蹤 – 灰色
- 擷取的追蹤 – 橘色
- 篩選的追蹤集 A – 綠色
- 篩選的追蹤集 B – 藍色

Example – 回應時間分佈

回應時間分發是顯示特定回應時間追蹤數的圖表。按一下並拖曳，在回應時間分發內選取範圍。這會選取特定回應時間內所有追蹤的 `responseTime` 工作追蹤集，並建立其篩選條件。

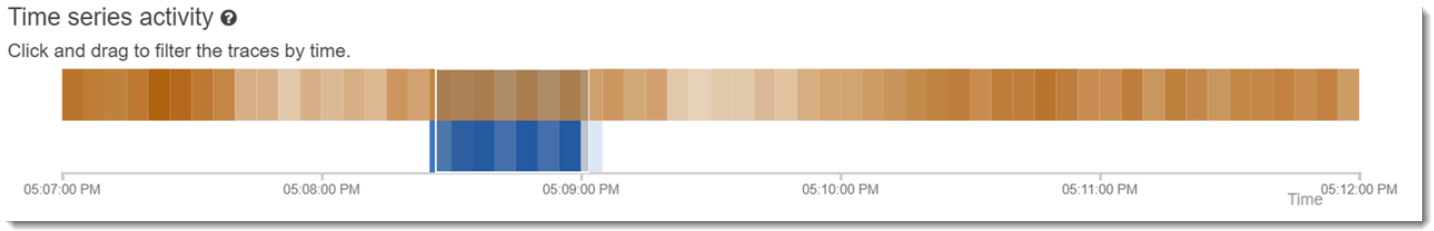


時間序列活動

時間序列活動圖會顯示特定期間的追蹤數目。顏色指標會鏡像回應時間分發的折線圖顏色。活動序列內的顏色區塊愈深愈滿，表示指定時間內的追蹤愈多。

Example – 時間序列活動

按一下並拖曳，在時間序列活動圖中選取範圍。這會選取特定時間範圍內所有追蹤的 `timerange` 工作追蹤集，並建立篩選條件。



工作流程範例

下列範例顯示 X-Ray Analytics 主控台的常見使用案例。每個範例都會示範主控台體驗的主要功能。因為是群組，所以範例會遵循基本的故障診斷工作流程。這些步驟會逐步解說如何先找出運作狀況不良節點，然後如何與 Analytics 主控台互動，以自動產生比較查詢。一旦透過查詢縮小範圍，您最終會看到特定追蹤的詳細資訊，以判斷是什麼損害伺服器的運作狀況。

觀察服務圖中的故障

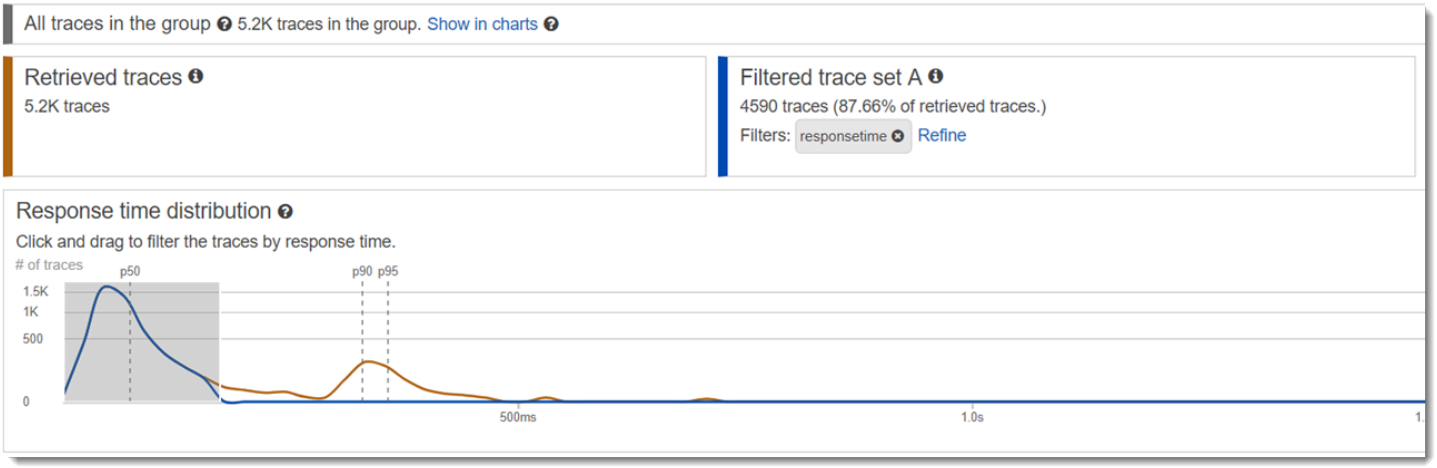
追蹤映射會根據成功呼叫與錯誤和錯誤的比例，將每個節點的運作狀態著色。當您在節點上看到紅色百分比時，表示有故障。使用 X-Ray Analytics 主控台進行調查。

如需如何讀取追蹤映射的詳細資訊，請參閱檢視追蹤映射。

識別回應時間高峰

使用回應時間分發，您可以觀察回應時間的高峰。只要選取回應時間的高峰，就會更新圖表下方的資料表，公開所有相關聯的指標，例如狀態碼。

當您按一下並拖曳時，X-Ray 會選取並建立篩選條件。它會以灰色陰影顯示在圖形線條的頂部。您現在可以沿著分佈左右拖曳陰影，更新選取範圍和篩選條件。



檢視有狀態碼標記的所有追蹤

您可以使用圖表下方的指標表，深入了解所選高峰的追蹤。按一下 HTTP STATUS CODE 表的資料列，即會自動建立工作資料集的篩選條件。例如，您可以檢視狀態碼為 500 的所有追蹤。這會在追蹤集圖標中建立名為 `http.status` 的篩選條件標籤。

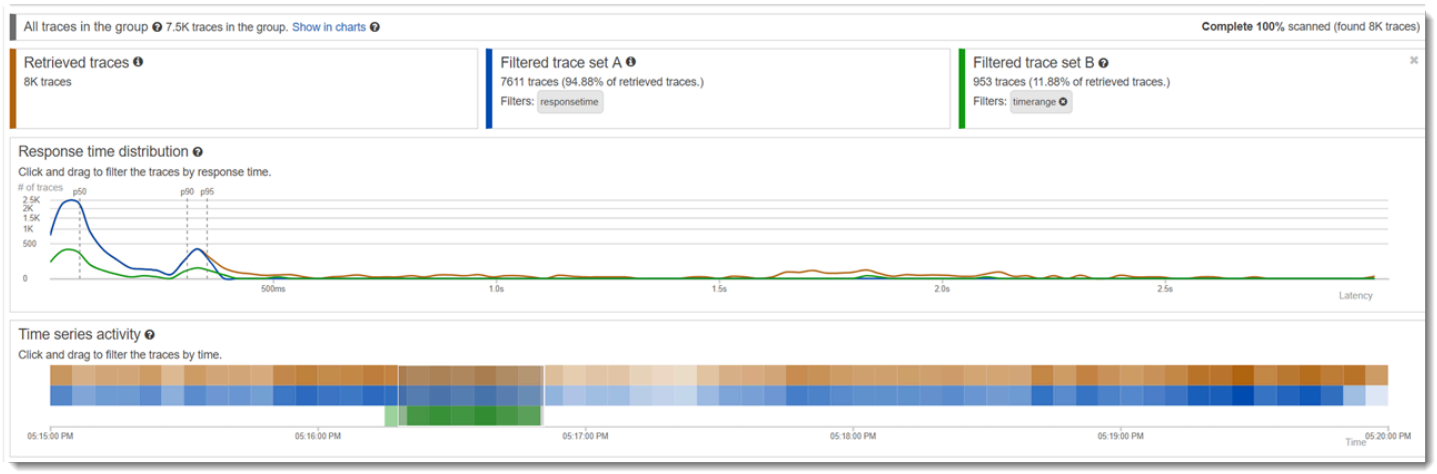
檢視在子群組中並與使用者相關聯的所有項目

深入了解以使用者、URL、回應時間根本原因或其他預先定義的屬性為基礎的錯誤集。例如，若要另行篩選狀態碼為 500 的追蹤集，請選取 USERS (使用者) 表中的資料列。這會讓追蹤集圖標出現兩個篩選條件標籤：之前已指定的`http.status` 以及 `user`。

比較兩個具有不同篩選標準的追蹤集

比較所有不同的使用者及其 POST 請求，尋找其他的差異和關聯性。套用您的第一組篩選條件。它們在回應時間分發中定義為藍色線條。然後選取 Compare (比較)。一開始，這會建立追蹤集 A 的篩選條件副本。

若要繼續，請定義一組新的篩選條件，套用到追蹤集 B。這第二組條件會以綠色線條表示。以下範例示範以藍綠顏色組合的不同線條。



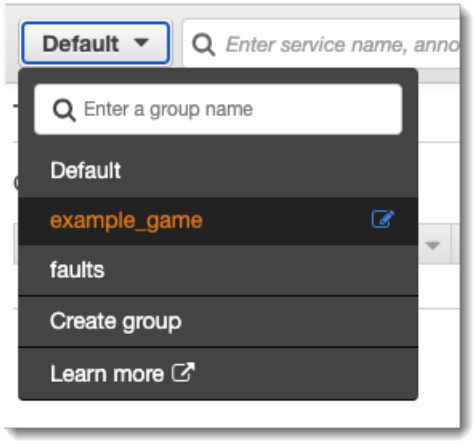
識別感興趣的追蹤以及檢視其詳細資訊

當您使用主控台篩選條件縮小範圍時，指標表下的追蹤清單就會變得更有意義。追蹤清單表將 URL、USER (使用者) 和 STATUS CODE (狀態碼) 的相關資訊結合到一份檢視中。如需更多詳情，請選取此表的資料列開啟追蹤的 detail (詳細資訊) 頁面，檢視其時間軸和原始資料。

設定 群組

群組是一系列追蹤，為篩選條件表達式所定義。您可以使用 群組來產生額外的服務圖表，並提供 Amazon CloudWatch 指標。您可以使用 AWS X-Ray 主控台或 X-Ray API 來建立和管理 服務的群組。本主題說明如何使用 X-Ray 主控台建立和管理群組。如需如何使用 X-Ray API 管理群組的資訊，請參閱 群組。

您可以建立追蹤映射、追蹤或分析的追蹤群組。當您建立群組時，群組會在所有三個頁面上的群組下拉式功能表上變成篩選條件：追蹤地圖、追蹤和分析。



群組會根據名稱或 Amazon Resource Name (ARN) 進行識別，且包含篩選條件表達式。此服務會比較傳入表達式的追蹤，並依序存放。如需如何建置篩選條件表達式的詳細資訊，請參閱 使用篩選條件表達式。

更新群組的篩選條件表達式不會變更已記錄的資料。更新僅適用於後續追蹤。這會導致圖表合併新舊表達式。若要避免這種情況，請刪除目前的群組並建立新的群組。

> **Note**
>
> 群組的計費方式是根據符合篩選條件表達式的擷取追蹤。如需詳細資訊，請參閱 AWS X-Ray 定價。

主題

建立群組

> **Note**
>
> 您現在可以從 Amazon CloudWatch 主控台設定 X-Ray 群組。您也可以繼續使用 X-Ray 主控台。

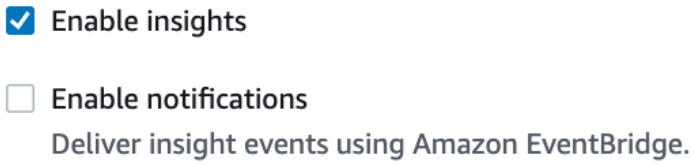
CloudWatch console

1. 登入 AWS Management Console ，並在 https：//https://console.aws.amazon.com/cloudwatch/ 開啟 CloudWatch 主控台。
2. 在左側導覽窗格中選擇設定。
3. 在 X-Ray 追蹤區段的群組下選擇檢視設定。
4. 選擇群組清單上方的建立群組。

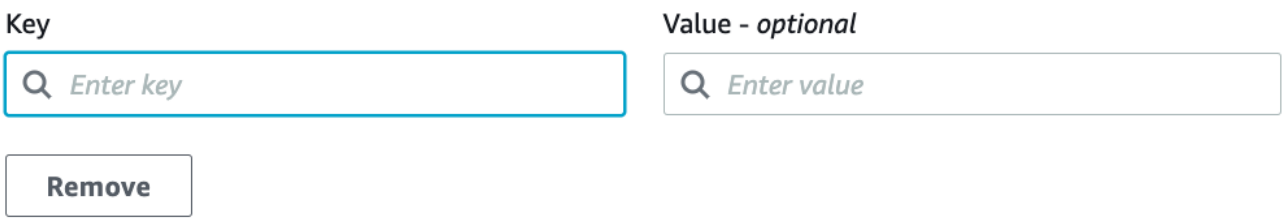
5. 在建立群組頁面上，輸入群組的名稱。群組名稱最多可有 32 個字元，且包含英數字元和破折號。群組名稱區分大小寫。
6. 輸入篩選條件表達式。如需如何建置篩選條件表達式的詳細資訊，請參閱 使用篩選條件表達式。在下列範例中， 群組會篩選來自服務 的故障追蹤`api.example.com`。 和 請求至回應時間大於或等於五秒的服務。

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

7. 在 Insights 中，啟用或停用群組的洞見存取。如需洞見的詳細資訊，請參閱使用 X-Ray 洞察。



8. 在標籤中，選擇新增標籤以輸入標籤索引鍵，也可以選擇輸入標籤值。繼續視需要新增其他標籤。標籤索引鍵必須是唯一的。若要刪除標籤，請選擇每個標籤下方的移除。如需標籤的詳細資訊，請參閱標記 X-Ray 取樣規則和群組。



9. 選擇建立群組。

X-Ray console

1. 登入 AWS Management Console 並開啟位於 https：//https://console.aws.amazon.com/xray/home. 的 X-Ray 主控台。
2. 從左側導覽窗格中的群組頁面，或從下列其中一個頁面上的群組選單開啟建立群組頁面：追蹤地圖、追蹤和分析。
3. 在建立群組頁面上，輸入群組的名稱。群組名稱最多可有 32 個字元，且包含英數字元和破折號。群組名稱區分大小寫。
4. 輸入篩選條件表達式。如需如何建置篩選條件表達式的詳細資訊，請參閱 使用篩選條件表達式。在下列範例中， 群組會篩選來自服務 的故障追蹤`api.example.com`。 和 請求至回應時間大於或等於五秒的服務。

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

5. 在 Insights 中，啟用或停用群組的洞見存取。如需洞見的詳細資訊，請參閱使用 X-Ray 洞察。

6. 在標籤中，輸入標籤索引鍵，並選擇性地輸入標籤值。當您新增標籤時，會顯示新行，供您輸入另一個標籤。標籤索引鍵必須是唯一的。若要刪除標籤，請選擇標籤列結尾的 X。如需標籤的詳細資訊，請參閱標記 X-Ray 取樣規則和群組。

7. 選擇建立群組。

套用群組

CloudWatch console

1. 登入 AWS Management Console ，並在 https：//https://console.aws.amazon.com/cloudwatch/ 開啟 CloudWatch 主控台。
2. 從 X-Ray 追蹤下的導覽窗格開啟下列其中一個頁面：
   - 追蹤映射
   - 追蹤
3. 在依 X-Ray 篩選群組篩選條件中輸入群組名稱。頁面上顯示的資料會變更為符合 群組中設定的篩選條件表達式。

X-Ray console

1. 登入 AWS Management Console ，並在 https：//https://console.aws.amazon.com/xray/home 開啟 X-Ray 主控台。
2. 從導覽窗格開啟下列其中一個頁面：

- 追蹤映射
- 追蹤
- 分析

3. 在群組功能表中，選擇您在 中建立的群組the section called “建立群組”。頁面上顯示的資料會變更為符合 群組中設定的篩選條件表達式。

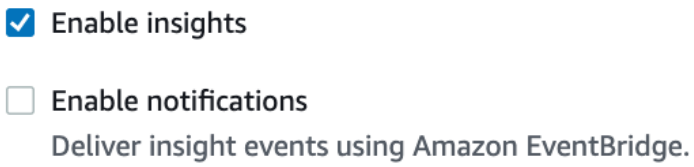
編輯群組

CloudWatch console

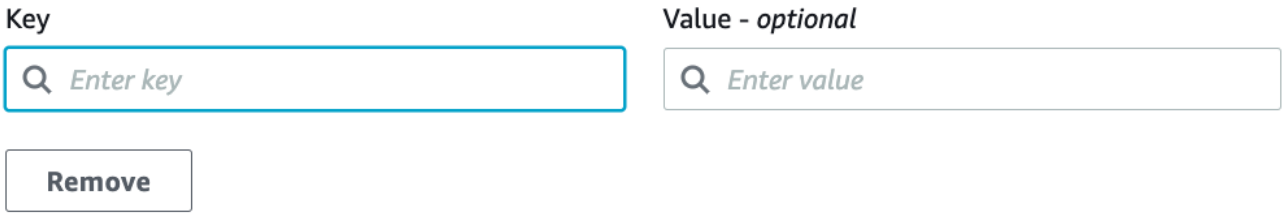
1. 登入 AWS Management Console ，並在 https://console.aws.amazon.com/cloudwatch/ 開啟 CloudWatch 主控台。
2. 在左側導覽窗格中選擇設定。
3. 在 X-Ray 追蹤區段的群組下選擇檢視設定。
4. 從群組區段中選擇群組，然後選擇編輯。
5. 雖然您無法重新命名群組，但您可以更新篩選條件表達式。如需如何建置篩選條件表達式的詳細資訊，請參閱 使用篩選條件表達式。在下列範例中， 群組會篩選來自 服務的故障追蹤`api.example.com`，其中請求 URL 地址包含 `example/game`，而請求的回應時間大於或等於五秒。

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

6. 在 Insights 中，啟用或停用群組的洞見存取。如需洞見的詳細資訊，請參閱使用 X-Ray 洞察。



7. 在標籤中，選擇新增標籤以輸入標籤索引鍵，也可以選擇輸入標籤值。繼續視需要新增其他標籤。標籤索引鍵必須是唯一的。若要刪除標籤，請選擇每個標籤下方的移除。如需標籤的詳細資訊，請參閱標記 X-Ray 取樣規則和群組。



8. 完成更新群組後，請選擇更新群組。

X-Ray console

1. 登入 AWS Management Console ，並在 https：//https://console.aws.amazon.com/xray/home 開啟 X-Ray 主控台。
2. 執行下列其中一項操作以開啟編輯群組頁面。

   a. 在群組頁面上，選擇要編輯的群組名稱。

   b. 在下列其中一個頁面上的群組選單上，指向群組，然後選擇編輯。

   - 追蹤映射
   - 追蹤
   - 分析

3. 雖然您無法重新命名群組，但您可以更新篩選條件表達式。如需如何建置篩選條件表達式的詳細資訊，請參閱 使用篩選條件表達式。在下列範例中， 群組會篩選來自 服務的故障追蹤`api.example.com`，其中請求 URL 地址包含 `example/game`，而請求的回應時間大於或等於五秒。

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

4. 在 Insights 中，啟用或停用群組的洞察和洞察通知。如需洞見的詳細資訊，請參閱使用 X-Ray 洞察。

5. 在標籤中，編輯標籤索引鍵和值。標籤索引鍵必須是唯一的。標籤值是選用的；如果需要，您可以刪除值。若要刪除標籤，請選擇標籤列結尾的 X。如需標籤的詳細資訊，請參閱標記 X-Ray 取樣規則和群組。

6. 完成更新群組後，請選擇更新群組。

複製群組

複製群組會建立新的群組，其中包含現有群組的篩選條件表達式和標籤。當您複製群組時，新群組的名稱與其複製的群組相同，並`-clone`附加到名稱。

CloudWatch console

1. 登入 AWS Management Console ，並在 https：//https://console.aws.amazon.com/cloudwatch/ 開啟 CloudWatch 主控台。
2. 在左側導覽窗格中選擇設定。
3. 在 X-Ray 追蹤區段的群組下選擇檢視設定。
4. 從群組區段中選擇群組，然後選擇複製。
5. 在建立群組頁面上，群組的名稱為 *`group-name`*`-clone`。或者，輸入群組的新名稱。群組名稱最多可有 32 個字元，且包含英數字元和破折號。群組名稱區分大小寫。
6. 您可以保留現有群組的篩選條件表達式，或選擇性地輸入新的篩選條件表達式。如需如何建置篩選條件表達式的詳細資訊，請參閱 使用篩選條件表達式。在下列範例中， 群組會篩選來自服務 的故障追蹤`api.example.com`。 和 請求至回應時間大於或等於五秒的服務。

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

7. 在標籤中，視需要編輯標籤索引鍵和值。標籤索引鍵必須是唯一的。標籤值是選用的；如果需要，您可以刪除值。若要刪除標籤，請選擇標籤列結尾的 X。如需標籤的詳細資訊，請參閱標記 X-Ray 取樣規則和群組。
8. 選擇建立群組。

X-Ray console

1. 登入 AWS Management Console ，並在 https：//https://console.aws.amazon.com/xray/home 開啟 X-Ray 主控台。
2. 從左側導覽窗格開啟群組頁面，然後選擇您要複製的群組名稱。
3. 從動作功能表中選擇複製群組。
4. 在建立群組頁面上，群組的名稱為 *group-name*-clone。或者，輸入群組的新名稱。群組名稱最多可有 32 個字元，且包含英數字元和破折號。群組名稱區分大小寫。
5. 您可以保留現有群組的篩選條件表達式，或選擇性地輸入新的篩選條件表達式。如需如何建置篩選條件表達式的詳細資訊，請參閱 使用篩選條件表達式。在下列範例中， 群組會篩選來自服務 的故障追蹤api.example.com。 和 請求至回應時間大於或等於五秒的服務。

   ```
   service("api.example.com") { fault = true OR responsetime >= 5 }
   ```
6. 在標籤中，視需要編輯標籤索引鍵和值。標籤索引鍵必須是唯一的。標籤值是選用的；如果需要，您可以刪除值。若要刪除標籤，請選擇標籤列結尾的 X。如需標籤的詳細資訊，請參閱標記 X-Ray 取樣規則和群組。
7. 選擇建立群組。

刪除群組

請依照本節中的步驟刪除群組。您無法刪除預設群組。

CloudWatch console

1. 登入 AWS Management Console ，並在 https：//https://console.aws.amazon.com/cloudwatch/ 開啟 CloudWatch 主控台。
2. 在左側導覽窗格中選擇設定。
3. 在 X-Ray 追蹤區段的群組下選擇檢視設定。
4. 從群組區段中選擇群組，然後選擇刪除。
5. 系統提示您確認時，請選擇刪除。

X-Ray console

1. 登入 AWS Management Console ，並在 https：//https://console.aws.amazon.com/xray/home 開啟 X-Ray 主控台。

2. 從左側導覽窗格開啟群組頁面，然後選擇您要刪除的群組名稱。
3. 在動作功能表中，選擇刪除群組。
4. 當您收到確認提示時，請選擇刪除。

在 Amazon CloudWatch 中檢視群組指標

建立群組之後，系統會針對群組的篩選條件表達式檢查傳入的追蹤，因為這些追蹤存放在 X-Ray 服務中。符合每個條件的追蹤數量指標每分鐘都會發佈至 Amazon CloudWatch。在編輯群組頁面上選擇檢視指標會開啟 CloudWatch 主控台至指標頁面。如需如何使用 CloudWatch 指標的詳細資訊，請參閱《Amazon CloudWatch 使用者指南》中的使用 Amazon CloudWatch 指標。 Amazon CloudWatch

CloudWatch console

1. 登入 AWS Management Console ，並在 https：//https://console.aws.amazon.com/cloudwatch/ 開啟 CloudWatch 主控台。
2. 在左側導覽窗格中選擇設定。
3. 在 X-Ray 追蹤區段的群組下選擇檢視設定。
4. 從群組區段中選擇群組，然後選擇編輯。
5. 在編輯群組頁面上，選擇檢視指標。

   CloudWatch 主控台指標頁面會在新標籤中開啟。

X-Ray console

1. 登入 AWS Management Console ，並在 https：//https://console.aws.amazon.com/xray/home 開啟 X-Ray 主控台。
2. 從左側導覽窗格開啟群組頁面，然後選擇您要檢視指標的群組名稱。
3. 在編輯群組頁面上，選擇檢視指標。

   CloudWatch 主控台指標頁面會在新標籤中開啟。

設定 取樣規則

您可以使用 AWS X-Ray 主控台來設定 服務的取樣規則。支援主動追蹤搭配取樣組態 AWS 服務 的 X-Ray SDK 和 使用取樣規則來決定要記錄的請求。

主題

設定 取樣規則

您可以針對下列使用案例設定取樣：

- API Gateway Entrypoint – API Gateway 支援取樣和主動追蹤。若要在 API 階段上啟用主動追蹤，請參閱 的 Amazon API Gateway 主動追蹤支援 AWS X-Ray。
- AWS AppSync – AWS AppSync 支援取樣和主動追蹤。若要啟用請求的 AWS AppSync 主動追蹤，請參閱使用 AWS X-Ray 追蹤。
- 在運算平台上檢測 X-Ray 開發套件 – 使用 Amazon EC2、Amazon ECS 或 等運算平台時 AWS Elastic Beanstalk，當應用程式已使用最新的 X-Ray 開發套件檢測時，支援取樣。

自訂抽樣規則

透過自訂抽樣規則，您可以控制記錄的資料量。您也可以修改抽樣行為，而無需修改或重新部署程式碼。取樣規則會告知 X-Ray SDK 要針對一組條件記錄多少請求。根據預設，X-Ray 開發套件每秒記錄第一個請求，以及任何其他請求的 5%。每秒一個請求是儲槽。這可確保只要服務持續提供請求，每秒都會記錄至少一個追蹤。5% 是超過儲槽大小的額外請求抽樣「速率」。

您可以設定 X-Ray 開發套件，從程式碼隨附的 JSON 文件讀取取樣規則。但是，當您執行服務的多個執行個體時，每個執行個體都會獨立執行抽樣。這會導致抽樣請求的整體百分比增加，因為所有執行個體的儲槽都會加在一起。此外，若要更新本機取樣規則，您必須重新部署程式碼。

透過在 X-Ray 主控台中定義取樣規則，並設定 SDK 從 X-Ray 服務讀取規則，您可以避免這兩個問題。服務會管理每個規則的儲槽，並根據執行中的執行個體數，將配額指派給每個服務執行個體來平均分散儲槽。儲槽限制是根據您設定的規則所計算。由於規則是在 服務中設定，因此您可以管理規則，而無需進行其他部署。

> **Note**
>
> X-Ray 在套用抽樣規則時採用最佳作法，在某些情況下，有效抽樣率可能不會完全符合設定的抽樣規則。不過，隨著時間的推移，抽樣的請求數量應該接近設定的百分比。

您現在可以從 Amazon CloudWatch 主控台內設定 X-Ray 取樣規則。您也可以繼續使用 X-Ray 主控台。

CloudWatch console

在 CloudWatch 主控台中設定抽樣規則

1. 登入 AWS Management Console ，並在 https://console.aws.amazon.com/cloudwatch/ ：// 開啟 CloudWatch 主控台。
2. 在左側導覽窗格中選擇設定。
3. 在 X-Ray 追蹤區段中選擇取樣規則下的檢視設定。
4. 若要建立規則，請選擇 Create sampling rule (建立抽樣規則)。

   若要編輯規則，請選擇規則，然後選擇編輯以進行編輯。

   若要刪除規則，請選擇規則，然後選擇刪除以刪除規則。

X-Ray console

在 X-Ray 主控台中設定取樣規則

1. 開啟 X-Ray 主控台。
2. 在左側導覽窗格中選擇取樣。
3. 若要建立規則，請選擇 Create sampling rule (建立抽樣規則)。

   若要編輯規則，請選擇規則的名稱。

   若要刪除規則，請選擇規則並使用 Actions (動作) 功能表來刪除它。

抽樣規則選項

下列選項可用於每個規則。字串值可以使用萬用字元來以符合單一字元 (?) 或零或多個字元 (*)。

抽樣規則選項

- 規則名稱（字串）– 規則的唯一名稱。
- 優先順序（介於 1 到 9999 之間的整數）– 取樣規則的優先順序。服務會以遞增的優先順序來評估規則，並使用符合的第一個規則來決定取樣決策。
- 儲存器（非負整數）– 套用固定速率之前，每秒要檢測的比對請求固定數量。儲槽不會直接用於服務，而是集體套用至使用該規則的所有服務。
- 速率（介於 0 到 100 之間的整數）– 儲槽用盡後，與檢測相符的請求百分比。在主控台中設定取樣規則時，請選擇介於 0 到 100 之間的百分比。使用 JSON 文件在用戶端 SDK 中設定取樣規則時，請提供介於 0 和 1 之間的百分比值。
- 服務名稱（字串）– 檢測服務的名稱，如追蹤映射中所示。
  - X-Ray SDK – 您在記錄器上設定的服務名稱。
  - Amazon API Gateway – *api-name*/*stage*。
- 服務類型（字串）– 顯示在追蹤映射中的服務類型。對於 X-Ray SDK，請套用適當的外掛程式來設定服務類型：
  - `AWS::ElasticBeanstalk::Environment` – AWS Elastic Beanstalk 環境（外掛程式）。
  - `AWS::EC2::Instance` – Amazon EC2 執行個體（外掛程式）。
  - `AWS::ECS::Container` – Amazon ECS 容器（外掛程式）。
  - `AWS::APIGateway::Stage` – Amazon API Gateway 階段。
  - `AWS::AppSync::GraphQLAPI` – AWS AppSync API 請求。
- 主機（字串）– 來自 HTTP 主機標頭的主機名稱。
- HTTP 方法（字串）– HTTP 請求的方法。
- URL 路徑（字串）– 請求的 URL 路徑。
  - X-Ray SDK – HTTP 請求 URL 的路徑部分。
- 資源 ARN（字串）– 執行服務之 AWS 資源的 ARN。
  - X-Ray SDK – 不支援。軟體開發套件僅能使用 Resource ARN (資源 ARN) 設為 * 的規則。
  - Amazon API Gateway – 階段 ARN。
- （選用）屬性（索引鍵和值）– 取樣決策時已知的區段屬性。
  - X-Ray SDK – 不支援。軟體開發套件會忽略指定屬性的規則。
  - Amazon API Gateway – 來自原始 HTTP 請求的標頭。

抽樣規則範例

Example – 預設規則，沒有儲槽且速率低

您可以修改預設規則的儲槽和速率。預設規則會套用至不符合任何其他規則的請求。

- 儲存庫： **0**
- 速率：**5**(**0.05**如果使用 JSON 文件設定）

Example – 偵錯規則以追蹤有問題路由的所有請求

會暫時套用高優先順序的規則來進行除錯。

- 規則名稱： **DEBUG – history updates**
- 優先順序： **1**
- 儲存庫： **1**
- 速率：**100**(**1**如果使用 JSON 文件設定）
- 服務名稱： **Scorekeep**
- Service type (服務類型)：*****
- 主機： *****
- HTTP 方法： **PUT**
- URL 路徑： **/history/***
- 資源 ARN： *****

Example – POSTs的較高最低速率

- 規則名稱： **POST minimum**
- 優先順序： **100**
- 儲存庫： **10**
- 速率：**10**(**.1**如果使用 JSON 文件設定）
- 服務名稱： *****
- Service type (服務類型)：*****
- 主機： *****

- HTTP 方法： **POST**
- URL 路徑： *****
- 資源 ARN： *****

將您的服務設定為使用抽樣規則

X-Ray SDK 需要額外的組態，才能使用您在主控台中設定的抽樣規則。如需設定抽樣策略的詳細資訊，請參閱適用您語言的組態主題：

- Java：抽樣規則
- Go： 抽樣規則
- Node.js： 抽樣規則
- Python：抽樣規則
- Ruby： 抽樣規則
- .NET： 抽樣規則

如需 API Gateway，請參閱 的 Amazon API Gateway 主動追蹤支援 AWS X-Ray。

檢視抽樣結果

X-Ray 主控台取樣頁面顯示服務如何使用每個取樣規則的詳細資訊。

Trend (趨勢) 資料行會顯示過去幾分鐘內使用規則的方式。每個資料行都會顯示 10 秒間的統計資料。

抽樣統計資料

- 符合的規則總數：符合此規則的請求數量。此數量不包含本來會和此規則相符，但是卻先符合高優先順序規則的請求。
- 取樣總數：記錄的請求數。
- 以固定速率取樣：套用規則固定速率取樣的請求數。
- 使用儲槽限制取樣：使用 X-Ray 指派的配額取樣的請求數量。
- 從儲槽借用：從儲槽借用取樣的請求數量。服務第一次將請求配對至規則時，X-Ray 尚未指派配額。不過，如果儲槽至少為 1，則服務會借用每秒一個追蹤，直到 X-Ray 指派配額為止。

如需抽樣統計資料和服務使用抽樣規則方式的詳細資訊，請參閱透過 X-Ray API 使用取樣規則。

後續步驟

您可以使用 X-Ray API 來管理抽樣規則。透過 API，您可以透過編寫程式設計的方式在排程上建立及更新規則，或是回應警示或通知。如需說明及其他規則範例，請參閱使用 AWS X-Ray API 設定抽樣、分組和加密設定。

X-Ray SDK 和 AWS 服務 也使用 X-Ray API 來讀取取樣規則、報告取樣結果，以及取得取樣目標。服務必須追蹤其套用每個規則的頻率、根據優先順序評估規則，以及在請求符合 X-Ray 尚未為服務指派配額的規則時從儲槽借用。如需服務使用 API 進行抽樣的詳細資訊，請參閱透過 X-Ray API 使用取樣規則。

當 X-Ray SDK 呼叫取樣 APIs時，它會使用 X-Ray 協助程式做為代理。若您已使用 TCP 連接埠 2000，現在您可以設定精靈，在不同的連接埠執行代理。如需詳細資訊，請參閱 設定 AWS X-Ray 協助程式。

主控台深層連結

您可以使用路由和查詢來深入連結至特定追蹤，或追蹤和追蹤映射的篩選檢視。

主控台頁面

- 歡迎頁面 – xray/home#/welcome
- 入門 – xray/home#/getting-started
- 追蹤映射 – xray/home#/service-map
- 追蹤 – xray/home#/traces

追蹤

您可以為個別追蹤的時間表檢視、原始檢視及映射檢視建立連結。

追蹤時間軸 – `xray/home#/traces/`*`trace-id`*

原始追蹤資料 – `xray/home#/traces/`*`trace-id`*`/raw`

Example – 原始追蹤資料

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

篩選條件表達式

連結到追蹤的篩選清單。

篩選的追蹤檢視 – xray/home#/traces?filter=*filter-expression*

Example – 篩選條件表達式

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

Example – 篩選條件表達式 (URL 編碼)

```
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
```

如需篩選條件表達式的詳細資訊，請參閱使用篩選條件表達式。

時間範圍

指定時間長度或開始及結束時間 (格式為 ISO8601)。時間範圍以 UTC 表示，最長可達 6 小時。

時間長度 – xray/home#/*page*?timeRange=*range-in-minutes*

Example – 過去一小時的追蹤映射

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

開始和結束時間 – xray/home#/*page*?timeRange=*start*~*end*

Example – 時間範圍精確到 秒

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

Example – 時間範圍精確到 分鐘

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

區域

指定 AWS 區域 以連結至該區域中的頁面。若您未指定區域，主控台會將您重新導向至最後一個前往的區域。

區域 – xray/home**?region=*region***#/*page*

Example – 美國西部 （ 奧勒岡 ） 的追蹤地圖 (us-west-2)

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

當您將區域與其他查詢參數一起包含時，區域查詢會在雜湊之前進行，而 X-Ray-specific查詢會在頁面名稱之後進行。

Example – 美國西部 （ 奧勒岡 ） (us-west-2) 中最後一個小時的追蹤映射

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

合併

Example – 具有持續時間篩選條件的最近追蹤

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

輸出

- 頁面 – 追蹤
- 時間範圍 – 過去 15 分鐘
- 篩選條件 – 持續時間 >= 5 且持續時間 <= 8

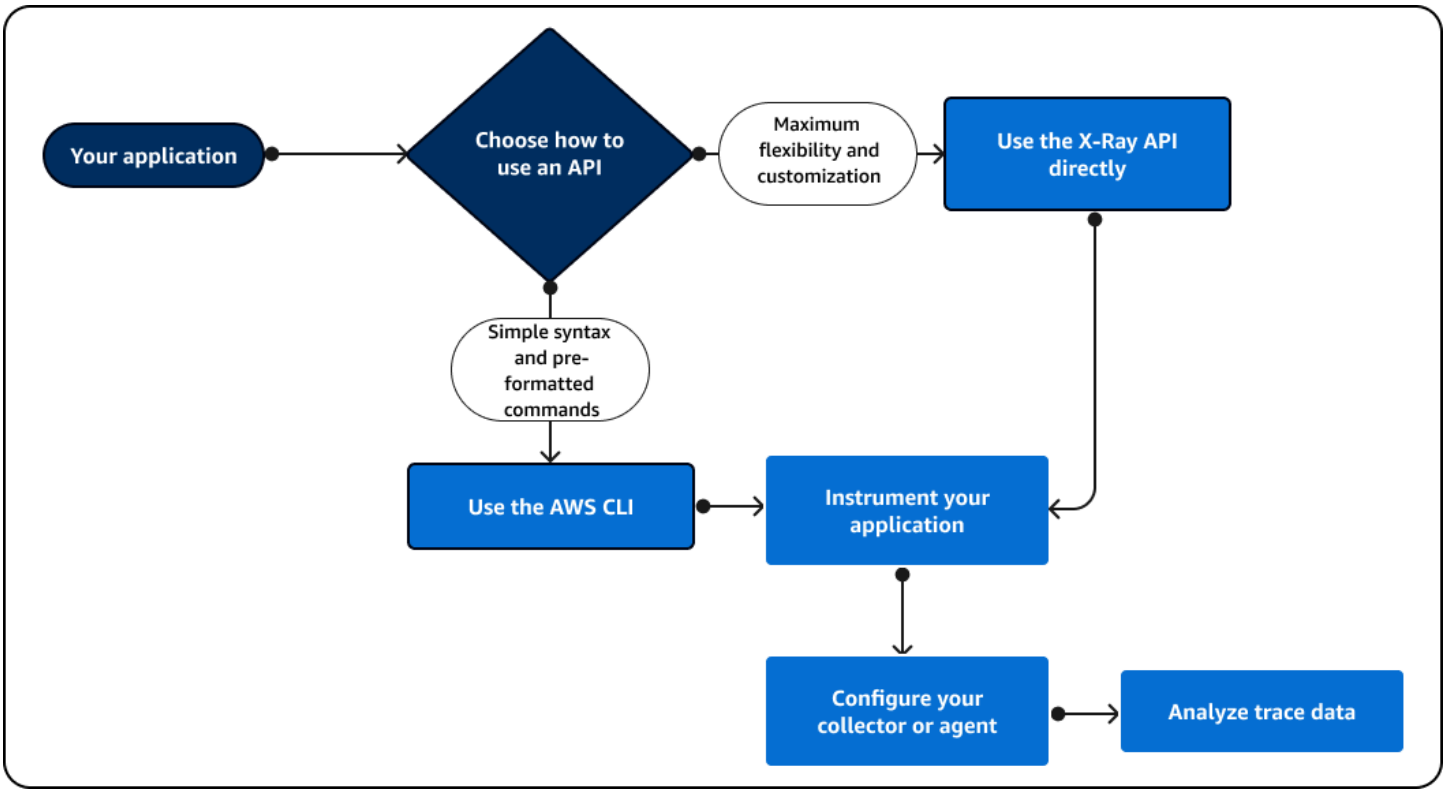
使用 X-Ray API

如果 X-Ray SDK 不支援您的程式設計語言，您可以直接使用 X-Ray APIs或 AWS Command Line Interface (AWS CLI) 來呼叫 X-Ray API 命令。使用下列指引來選擇您與 API 的互動方式：

- 使用 使用預先格式化的命令或 請求中的選項 AWS CLI ，來使用更簡單的語法。
- 直接使用 X-Ray API，以針對您對 X-Ray 提出的請求，獲得最大的彈性和自訂。

如果您直接使用 X-Ray API 而非 AWS CLI，則必須以正確的資料格式來參數化請求，也可能必須設定身分驗證和錯誤處理。

下圖顯示選擇如何與 X-Ray API 互動的指引：



使用 X-Ray API 將追蹤資料直接傳送到 X-Ray。X-Ray API 支援 X-Ray SDK 中提供的所有函數，包括下列常見動作：

- PutTraceSegments – 將區段文件上傳至 X-Ray。
- BatchGetTraces – 擷取追蹤 IDs 清單中的追蹤清單。每個擷取的追蹤都是來自單一請求的區段文件集合。
- GetTraceSummaries – 擷取追蹤IDs 和註釋。您可以指定 `FilterExpression`來擷取追蹤摘要的子集。
- GetTraceGraph – 擷取特定追蹤 ID 的服務圖表。
- GetServiceGraph – 擷取JSON格式化文件，描述處理傳入請求和呼叫下游請求的服務。

您也可以使用應用程式程式碼中的 AWS Command Line Interface (AWS CLI)，以程式設計方式與 X-Ray 互動。 AWS CLI 支援 X-Ray SDK 中提供的所有函數，包括其他函數 AWS 服務。下列函數是先前以更簡單格式列出的 API 操作版本：

- put-trace-segments – 將區段文件上傳至 X-Ray。
- batch-get-traces – 擷取追蹤 IDs 清單中的追蹤清單。每個擷取的追蹤都是來自單一請求的區段文件集合。
- get-trace-summaries – 擷取追蹤IDs 和註釋。您可以指定 `FilterExpression`來擷取追蹤摘要的子集。
- get-trace-graph – 擷取特定追蹤 ID 的服務圖表。
- get-service-graph – 擷取`JSON`格式化文件，描述處理傳入請求和呼叫下游請求的服務。

若要開始使用，您必須AWS CLI為作業系統安裝 。 AWS 支援 Linux、 macOS和 Windows 作業系統。如需 X-Ray 命令清單的詳細資訊，請參閱適用於 AWS CLI X-Ray 的命令參考指南。

主題

搭配 CLI AWS 使用 AWS X-Ray API

CLI AWS 可讓您直接存取 X-Ray 服務，並使用 X-Ray 主控台用來擷取服務圖表和原始追蹤資料的相同 APIs。範例應用程式包含指令碼，示範如何將這些 APIs與 CLI AWS 搭配使用。

先決條件

此教學使用 Scorekeep 範例應用程式和隨附的指令碼，以產生追蹤資料和服務地圖。按照入門教學中的指示啟動應用程式。

本教學課程使用 AWS CLI 來顯示 X-Ray API 的基本使用方式。適用於 Windows、Linux 和 OS-X 的 AWS CLI 提供所有人公有 APIs的命令列存取 AWS 服務。

> **Note**
>
> 您必須驗證您的 AWS CLI 已設定為與 Scorekeep 範例應用程式建立所在的相同區域。

隨附的指令碼 (用來測試範例應用程式) 會使用 cURL，將流量傳送到 API 和 jq 以剖析輸出。您可以從 jq stedolan.github.io 下載可執行檔，並從 下載curl可執行檔https://curl.haxx.se/download.html。大多數的 Linux 和 OS X 安裝都包括 cURL。

產生追蹤資料

當遊戲進行中時，Web 應用程式會每隔幾秒鐘持續產生對 API 的流量，但只會產生一種類型的請求。使用 test-api.sh 指令碼來執行端對端案例，並在您測試 API 時產生更多元化的追蹤資料。

使用 **test-api.sh** 指令碼

1. 開啟 Elastic Beanstalk 主控台。
2. 導覽至您環境的管理主控台。
3. 複製頁面標頭的環境 URL。
4. 開啟 bin/test-api.sh 並將 API 的值取代為您環境的 URL。

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

5. 執行指令碼來產生對 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

CLI AWS 為 X-Ray 提供的所有 API 動作提供命令，包括 GetServiceGraph和 GetTraceSummaries。如需所使用的所有支援動作和資料類型詳細資訊，請參閱 AWS X-Ray API 參考。

Example bin/service-graph.sh

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

指令碼會擷取最後 10 分鐘的服務圖表。

```
~/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'
```

指令碼會擷取前一分鐘和兩個分鐘之間產生的追蹤 URL。

```
~/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[*]'
```

指令碼會擷取前一分鐘和兩個分鐘之間產生的完整追蹤。

```
~/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
    }
...
```


清除

終止您的 Elastic Beanstalk 環境以關閉 Amazon EC2 執行個體、DynamoDB 資料表和其他資源。

若要終止您的 Elastic Beanstalk 環境

1. 開啟 Elastic Beanstalk 主控台。
2. 導覽至您環境的管理主控台。
3. 選擇動作。
4. 選擇 Terminate Environment (終止環境)。
5. 選擇終止。

追蹤資料會在 30 天後自動從 X-Ray 刪除。

傳送追蹤資料至 AWS X-Ray

您可以將追蹤資料以區段文件的形式傳送至 X-Ray。區段文件是一種 JSON 格式字串，包含您應用程式在處理請求時執行工作的相關資訊。您的應用程式可記錄其在區段中自行執行的工作相關資料，或是在子區段中使用下游服務和資源的工作相關資料。

區段會記錄您應用程式執行工作的相關資訊。區段最少會記錄其在任務上耗費的時間、名稱及兩個 ID。追蹤 ID 會在請求於服務間傳送時追蹤請求。區段 ID 會追蹤單一服務為請求完成的工作。

Example 最小的完成區段

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

接收到請求時，您可以傳送正在進行中的區段做為預留位置，直到完成請求。

Example 進行中的區段

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

您可以直接透過 PutTraceSegments或透過 X-Ray 協助程式將區段傳送至 X-Ray。

大多數應用程式會呼叫其他 服務或存取使用 AWS SDK 的資源。在子區段中記錄下游呼叫的相關資訊。X-Ray 使用子區段來識別未傳送區段的下游服務，並在服務圖表上為其建立項目。

子區段可內嵌在完整區段文件中，或是分別傳送。個別傳送子區段，以非同步追蹤長時間執行請求的下游呼叫，或避免超過區段文件大小上限 (64 kB)。

Example 子區段

子區段具備 `subsegment` 的 `type`，以及可識別父區段的 `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”
}
```

如需您可以包含在區段和子區段中欄位和值的詳細資訊，請參閱 AWS X-Ray 區段文件。

章節

- 產生追蹤 ID
- 使用 PutTraceSegments
- 將區段文件傳送至 X-Ray 協助程式

產生追蹤 ID

若要將資料傳送至 X-Ray，您必須為每個請求產生唯一的追蹤 ID。

X-Ray 追蹤 ID 格式

X-Ray `trace_id`由以連字號分隔的三個數字組成。例如：`1-58406520-a006649127e371903a2de979`。其中包含：

- 版本編號，即 1。
- 使用 8 個十六進位數字的 Unix epoch 時間原始請求的時間。

  例如，10：00AM 2016 年 12 月 1 日 PST 以 epoch `58406520` 為單位是`1480615200`秒或十六進位數字。
- 追蹤的全域唯一 96 位元識別符，以 24 十六進位數字表示。

> **Note**
>
> X-Ray 現在支援使用 OpenTelemetry 建立IDs，以及符合 W3C 追蹤內容規格的任何其他架構。W3C 追蹤 ID 在傳送至 X-Ray 時，必須以 X-Ray 追蹤 ID 格式格式化。例如，W3C 追蹤 ID 的格式`4efaaf4d1e8720b39541901950019ee5`應該如同傳送至 X-Ray

1-4efaaf4d-1e8720b39541901950019ee5時一樣。X-Ray IDs 包含以 Unix epoch 時間表示的原始請求時間戳記，但在以 X-Ray 格式傳送 W3C 追蹤 IDs 時不需要。

您可以撰寫指令碼來產生 X-Ray IDs 以供測試。以下是兩個範例。

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"
```

如需建立追蹤 IDs並將區段傳送至 X-Ray 協助程式的指令碼，請參閱 Scorekeep 範例應用程式。

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

使用 PutTraceSegments

您可以使用 PutTraceSegments API 上傳區段文件。API 具備單一參數 (TraceSegmentDocuments)，會接受 JSON 區段文件清單。

使用 AWS CLI，使用 aws xray put-trace-segments命令將區段文件直接傳送至 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 命令處理程式和 Windows PowerShell 針對在 JSON 字串中引述和逸出引述具有不同需求。如需詳細資訊，請參閱 AWS CLI 《 使用者指南》中的引號字串。

輸出會列出任何處理失敗的區段。例如，若追蹤 ID 內的日期為距離現在太遠的過去日期，您可能會看到如下的錯誤。

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

您可以同時傳遞多個區段文件，並以空格區隔。

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

將區段文件傳送至 X-Ray 協助程式

您可以傳送區段和子區段到 X-Ray 協助程式，而不是將區段文件傳送到 X-Ray 協助程式，這會緩衝它們並批次上傳至 X-Ray API。X-Ray SDK 會將區段文件傳送至協助程式，以避免直接呼叫 AWS 。

> **Note**
>
> 如需執行精靈的說明，請參閱在本機執行 X-Ray 協助程式。

透過 UDP 連接埠 2000 以 JSON 傳送區段，並在前面加上精靈標頭 (`{"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，您可以從 Bash 終端機將區段文件傳送到精靈。將標頭和區段文件儲存到文字檔，並使用 `cat` 將它輸送到 `/dev/udp`。

```
$ 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"}
```

檢查協助程式日誌，確認已將區段傳送至 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)
```

從 取得資料 AWS X-Ray

AWS X-Ray 會處理您傳送給它的追蹤資料，以在 JSON 中產生完整的追蹤、追蹤摘要和服務圖表。您可以使用 CLI 直接從 API AWS 擷取產生的資料。

章節

擷取服務圖表

您可以使用 `GetServiceGraph` API 擷取 JSON 服務圖表。API 需要開始時間及結束時間。您可以從 Linux 終端機使用 `date` 命令來計算該時間。

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

date +%s 會印出日期 (秒)。使用此數字做為結束時間，並減去時間來取得開始時間。

Example 擷取最後 10 分鐘服務圖表的指令碼

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

下列範例顯示具有 4 個節點的服務圖表，包括用戶端節點、EC2 執行個體、DynamoDB 資料表和 Amazon SNS 主題。

Example GetServiceGraph 輸出

```
{
    "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
                }
            ]
        }
    ]
}
```

根據群組擷取服務圖表

若要根據群組的內容呼叫服務圖形，請包含 groupName 或 groupARN。以下範例為服務圖表呼叫名為 Example1 的群組。

Example 根據名稱為 Example1 群組擷取服務圖表的指令碼

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

擷取追蹤

您可以使用 GetTraceSummaries API 取得追蹤摘要清單。追蹤摘要包含您可以用來識別要完整下載追蹤的資訊，包含標註、請求和回應資訊，以及 ID。

呼叫 aws xray get-trace-summaries 時有兩個 TimeRangeType 標記可用：

- TraceId – 預設GetTraceSummaries搜尋使用 TraceID 時間，並傳回在計算[start_time, end_time)範圍內啟動的追蹤。此時間戳記範圍是根據 TraceId 中的時間戳記編碼計算，也可以手動定義。
- 事件時間 – 為了在一段時間內搜尋事件， AWS X-Ray 允許使用事件時間戳記搜尋追蹤。事件時間會傳回 [start_time, end_time) 範圍內作用中的追蹤，無論追蹤由何時開始。

使用 aws xray get-trace-summaries 命令來取得追蹤摘要清單。下列命令會使用預設的 TraceId 時間，取得 1 到 2 分鐘之間的追蹤摘要清單。

Example 取得追蹤摘要的指令碼

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

Example GetTraceSummaries 輸出

```
{
    "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
}
```

使用來自輸出的追蹤 ID 來透過 [BatchGetTraces](#) API 擷取完整追蹤。

Example BatchGetTraces 命令

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

Example BatchGetTraces 輸出


```
{
    "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": []
```

```
}
```

完整追蹤包含每個區段的文件，該文件是從所有使用相同追蹤 ID 接收到的區段文件編譯而成。這些文件不代表您的應用程式傳送到 X-Ray 的資料。而是代表 X-Ray 服務產生的已處理文件。X-Ray 會編譯應用程式傳送的區段文件，並移除不符合區段文件結構描述的資料，以建立完整的追蹤文件。

X-Ray 也會建立推斷區段，以便對不會自行傳送區段的服務進行下游呼叫。例如，當您使用經檢測的用戶端呼叫 DynamoDB 時，X-Ray SDK 會記錄子區段，其中包含從其觀點來看的呼叫詳細資訊。不過，DynamoDB 不會傳送對應的區段。X-Ray 會使用子區段中的資訊來建立推斷區段，以代表追蹤映射中的 DynamoDB 資源，並將其新增至追蹤文件。

若要從 API 取得多個追蹤，您需要追蹤 IDs清單，您可以使用 `get-trace-summaries` AWS CLI 查詢從 輸出擷取。將清單重新導向至 的輸入`batch-get-traces`，以取得特定期間內的完整追蹤。

Example 取得一分鐘期間完整追蹤的指令碼

```
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[*]'
```

擷取和精簡根本原因分析

使用 GetTraceSummaries API 產生追蹤摘要時，可以其 JSON 格式重複使用部分追蹤摘要，以根本原因為基礎建立精簡的篩選條件表達式。如需精簡步驟的演練，請參閱以下範例。

Example 範例 GetTraceSummaries 輸出 - 回應時間根本原因一節

```
{
  "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
        }
      ]
    }
  ]
}
```

透過編輯和忽略上述輸出，可利用此 JSON 篩選條件符合根本原因的實體。針對出現在 JSON 中的每一個欄位，任何待選項目必須完全相符，否則不會傳回追蹤。已移除的欄位會成為萬用字元值，與篩選條件表達式查詢結構相容的格式。

Example 重新格式化回應時間根本原因

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

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

然後，此 JSON 會透過呼叫 `rootcause.json = #[{}]`，做為篩選條件表達式的一部分使用。如需查詢篩選條件表達式的詳細資訊，請參閱篩選條件表達式一章。

Example 範例 JSON 篩選條件

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

使用 AWS X-Ray API 設定抽樣、分組和加密設定

AWS X-Ray 提供用於設定取樣規則、群組規則和加密設定的 APIs。

章節

- 加密設定
- 抽樣規則
- 群組

加密設定

使用 `PutEncryptionConfig`指定用於加密的 AWS Key Management Service (AWS KMS) 金鑰。

> **Note**
>
> X-Ray 不支援非對稱 KMS 金鑰。

```
$ 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"
    }
}
```

針對金鑰 ID，您可以使用別名 (如範例中所示)、金鑰 ID 或 Amazon Resource Name (ARN)。

使用 GetEncryptionConfig 以取得目前的組態。當 X-Ray 完成套用您的設定時，狀態會從 變更為 UPDATING 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"
    }
}
```

若要停止使用 KMS 金鑰並使用預設加密，請將加密類型設定為 NONE。

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

抽樣規則

您可以使用 X-Ray API 管理帳戶中的抽樣規則。如需新增和管理標籤的詳細資訊，請參閱 標記 X-Ray 取樣規則和群組。

使用 GetSamplingRules 取得所有抽樣規則。

```
$ 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
        }
    ]
}
```

預設規則會套用至不符合其他規則的所有請求。這是優先順序最低的規則，且無法刪除。不過，您可以使用 UpdateSamplingRule 變更速率和儲槽大小。

Example 的 API 輸入 **UpdateSamplingRule** – 10000-default.json

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

以下範例使用之前的檔案做為輸入，將預設規則變更為 1%、無儲槽。標籤是選擇性的。如果您選擇新增標籤，則需要標籤索引鍵，標籤值則為選用。若要從抽樣規則中移除現有標籤，請使用 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
        },
```

使用 `CreateSamplingRule` 建立額外的抽樣規則。當您建立規則時，大多數規則欄位都必須填寫。以下範例將建立兩個規則。第一個規則會設定 Scorekeep 範例應用程式的基本速率。此規則適用於所有 API 提供但不符合更高優先順序之規則的請求。

Example 適用於 **`UpdateSamplingRule`** – 9000-base-scorekeep.json 的 API 輸入

```
{
    "SamplingRule": {
        "RuleName": "base-scorekeep",
        "ResourceARN": "*",
        "Priority": 9000,
        "FixedRate": 0.1,
        "ReservoirSize": 5,
        "ServiceName": "Scorekeep",
        "ServiceType": "*",
        "Host": "*",
        "HTTPMethod": "*",
        "URLPath": "*",
        "Version": 1
    }
}
```

第二個規則也會套用至 Scorekeep，但它的優先順序更高且更具體。此規則會設定非常低的抽樣速率以輪詢請求。這些是用戶端每隔幾秒發出的 GET 請求，以檢查遊戲狀態的變更。

Example 的 API 輸入 **UpdateSamplingRule** – 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
    }
}
```

標籤是選擇性的。如果您選擇新增標籤，則需要標籤索引鍵，標籤值則為選用。

```
$ 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
    }
}
```

若要刪除抽樣規則，請使用 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
    }
}
```

群組

您可以使用 X-Ray API 來管理帳戶中的群組。群組是一系列追蹤，為篩選條件表達式所定義。您可以使用 群組來產生額外的服務圖表，並提供 Amazon CloudWatch 指標。從 取得資料 AWS X-Ray 如需透過 X-Ray API 使用服務圖表和指標的詳細資訊，請參閱 。如需 群組的詳細資訊，請參閱 設定 群組。如需新增和管理標籤的詳細資訊，請參閱 標記 X-Ray 取樣規則和群組。

使用 CreateGroup 建立群組 標籤是選擇性的。如果您選擇新增標籤，則需要標籤索引鍵，標籤值則為選用。

```
$ 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}"
}
```

使用 GetGroups 取得所有現有群組。

```
$ 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"
}
```

使用 UpdateGroup 更新群組 標籤是選擇性的。如果您選擇新增標籤，則需要標籤索引鍵，標籤值則為選用。若要從群組中移除現有標籤，請使用 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}"
}
```

用 DeleteGroup 刪除群組。

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

透過 X-Ray API 使用取樣規則

SDK AWS X-Ray 使用 X-Ray API 來取得取樣規則、報告取樣結果，以及取得配額。您可以使用這些 APIs 來進一步了解抽樣規則的運作方式，或以 X-Ray SDK 不支援的語言實作抽樣。

從使用 GetSamplingRules 取得所有抽樣規則開始。

```
$ 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
        }
    ]
}
```

輸出會包含預設規則和自訂規則。若您尚未建立抽樣規則，請參閱抽樣規則。

依照遞增優先順序排序，針對傳入請求評估規則。當規則相符時，使用固定速率和儲槽大小來進行抽樣決策。記錄抽樣請求並忽略 (針對追蹤用途) 未抽樣請求。在完成抽樣決策後停止評估規則。

規則儲槽大小是在套用固定速率前，每秒要記錄的追蹤目標數。儲槽會累積套用到所有服務，因此您無法直接使用它。不過，如果不是零，您可以在 X-Ray 指派配額之前，從儲槽借用每秒一個追蹤。在接收配額前，記錄每秒的第一個請求，然後將固定速率套用到其他請求。固定速率是介於 0 和 1.00 (100%) 間的十進位數。

以下範例會顯示對 GetSamplingTargets 發出的呼叫，其中包含在過去 10 秒間進行的抽樣決策詳細資訊。

```
$ 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": []
}
```

X-Ray 的回應包含要使用的配額，而不是從儲槽借用。在此範例中，服務在 10 秒間從儲槽借用 10 個追蹤，並將 10% 的固定速率套用到其他 100 個請求，其結果合計為 20 個抽樣請求。配額在五分鐘內（以存留時間表示）或直到指派新的配額為止都很有效。X-Ray 也可能指派比預設值更長的報告間隔，雖然不是在這裡。

> **Note**
>
> 第一次呼叫時，X-Ray 的回應可能不會包含配額。繼續從儲存借用，直到您獲得指派配額。

回應中的其他兩個欄位可能會指出輸入的問題。針對您最後一次呼叫 `GetSamplingRules` 檢查 `LastRuleModification`。若其較新，請取得規則的新複本。`UnprocessedStatistics` 可包含錯誤，指出規則已遭到刪除、輸入中的統計文件過舊，或是許可錯誤。

AWS X-Ray 區段文件

追蹤區段是您應用程式所處理請求的 JSON 表示。追蹤區段會記錄原始請求的相關資訊、應用程式在本機執行之工作的相關資訊，以及包含應用程式對 AWS 資源、HTTP APIs 和 SQL 資料庫進行下游呼叫相關資訊的子區段。

區段文件會將區段的相關資訊傳遞給 X-Ray。區段文件最多可達 64 kB，並包含具有子區段的整個區段、表示請求正在進行中的區段片段，或個別傳送的單一子區段。您可以使用 `PutTraceSegments` API 將客群文件直接傳送到 X-Ray。

X-Ray 會編譯和處理區段文件，以產生可查詢的追蹤摘要和完整追蹤，您可以分別使用 `GetTraceSummaries`和 `BatchGetTraces` APIs 存取。除了您傳送至 X-Ray 的區段和子區段之外，服務也會使用子區段中的資訊來產生推斷區段，並將其新增至完整追蹤。推斷區段代表追蹤地圖中的下游服務和資源。

X-Ray 提供區段文件的 JSON 結構描述。您可以在此處下載結構描述：xray-segmentdocument-schema-v1.0.0。以下章節會更詳細的說明結構描述中所列出的欄位和物件。

區段欄位的子集由 X-Ray 編製索引，以便與篩選條件表達式搭配使用。例如，如果您將區段上的 `user` 欄位設定為唯一識別符，您可以在 X-Ray 主控台中或使用 `GetTraceSummaries` API 搜尋與特定使用者相關聯的區段。如需詳細資訊，請參閱使用篩選條件表達式。

當您使用 X-Ray 開發套件檢測應用程式時，開發套件會為您產生區段文件。SDK 不會直接將區段文件傳送到 X-Ray，而是透過本機 UDP 連接埠將文件傳輸至 X-Ray 協助程式。如需詳細資訊，請參閱將區段文件傳送至 X-Ray 協助程式。

章節

區段欄位

區段會記錄您應用程式所處理請求的相關追蹤資訊。區段最少會記錄請求的名稱、ID、開始時間、追蹤 ID 及結束時間。

Example 最小的完成區段

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

以下欄位為區段的必要項目或條件式必要項目。

> **Note**
>
> 除非另有說明，否則值必須為字串 (最多 250 個字元)。

必要的區段欄位

- `name` – 處理請求的服務邏輯名稱，最多 200 個字元。例如，您應用程式的名稱或網域名稱。名稱可包含 Unicode 字母、數字、空格和下列符號：`_`、`.`、`:`、`/`、`%`、`&`、`#`、`=`、`+`、`\`、`-`、`@`
- `id` – 區段的 64 位元識別符，在相同追蹤中的區段中是唯一的，以 16 個十六進位數字表示。
- `trace_id` – 唯一識別符，可連接來自單一用戶端請求的所有區段和子區段。

  X-Ray 追蹤 ID 格式

  X-Ray `trace_id`由以連字號分隔的三個數字組成。例如：`1-58406520-a006649127e371903a2de979`。其中包含：

  - 版本編號，即 `1`。
  - 使用 8 個十六進位數字的 Unix epoch 時間原始請求的時間。

    例如，10：00AM 2016 年 12 月 1 日 PST 以 epoch `58406520` 為單位是`1480615200`秒或十六進位數字。
  - 追蹤的全域唯一 96 位元識別符，以 24 個十六進位數字表示。

> **Note**
>
> X-Ray 現在支援使用 OpenTelemetry 建立IDs，以及符合 W3C 追蹤內容規格的任何其他架構。W3C 追蹤 ID 在傳送至 X-Ray 時，必須以 X-Ray 追蹤 ID 格式格式化。例如，W3C 追蹤 ID 的格式`4efaaf4d1e8720b39541901950019ee5`應該如同傳送至 X-Ray `1-4efaaf4d-1e8720b39541901950019ee5`時一樣。X-Ray IDs 包含以 Unix epoch 時間表示的原始請求時間戳記，但在以 X-Ray 格式傳送 W3C 追蹤 IDs 時不需要。

> **追蹤 ID 安全**
>
> 您可在回應標頭中取得追蹤 ID。使用安全的隨機演算法產生追蹤 ID，以確保攻擊者無法計算未來的追蹤 ID，並使用這些 ID 傳送請求到您的應用程式。

- `start_time` – 區段建立時間的數字，以 epoch 時間的浮點秒為單位。例如 `1480615200.010` 或 `1.480615200010E9`。視需要使用任意小數位數。建議盡可能使用毫秒解析度。
- `end_time` – 區段關閉時間的數字。例如 `1480615200.090` 或 `1.480615200090E9`。指定 `end_time` 或 `in_progress`。
- `in_progress` – 布林值，設定為 `true`，而不是指定 `end_time`來記錄區段已啟動，但未完成。應用程式收到的請求需要很長時間時，請傳送進行中區段，以追蹤請求的接收情況。回應已傳送時，請傳送完成區段以覆寫進行中的區段。針對每個請求，請只傳送一個完成區段，以及一或零個進行中區段。

> **服務名稱**
>
> 區段的 `name`應與產生區段之服務的網域名稱或邏輯名稱相符。不過，這不會強制執行。具有許可的任何應用程式`PutTraceSegments`都可以使用任何名稱傳送客群。

以下欄位是區段的選擇性欄位。

選擇性區段欄位

- `service` – 包含應用程式相關資訊的物件。
  - `version` – 識別提供請求之應用程式版本的字串。
- `user` – 識別傳送請求之使用者的字串。

- `origin` – 執行您應用程式 AWS 的資源類型。

  支援值

  - `AWS::EC2::Instance` – Amazon EC2 執行個體。
  - `AWS::ECS::Container` – Amazon ECS 容器。
  - `AWS::ElasticBeanstalk::Environment` – Elastic Beanstalk 環境。

  當有多個值適用您的應用程式時，請使用最具指定性的。例如，多容器 Docker Elastic Beanstalk 環境會在 Amazon ECS 容器上執行您的應用程式，而該容器又會在 Amazon EC2 執行個體上執行。在此案例中，您會將來源設為 `AWS::ElasticBeanstalk::Environment`，因為環境是其他兩個資源的父系。
- `parent_id` – 您指定的子區段 ID，如果請求來自經檢測的應用程式。X-Ray SDK 會將父子區段 ID 新增至下游 HTTP 呼叫的追蹤標頭。在巢狀子區段的情況下，子區段可以有一個區段或子區段作為其父項。
- `http` – `http` 物件，其中包含原始 HTTP 請求的相關資訊。
- `aws` – `aws` 物件，其中包含應用程式提供請求之 AWS 資源的相關資訊。
- `error`、`fault`、 `throttle`和 `cause` – 錯誤???欄位，指出發生錯誤，並包含導致錯誤的例外狀況相關資訊。
- `annotations` – 具有索引鍵/值對的`annotations`物件，您希望 X-Ray 為搜尋編製索引。
- `metadata` – `metadata` 物件，其中包含您要存放在區段的任何其他資料。
- `subsegments` – `subsegment` 物件陣列。

子區段

您可以建立子區段來記錄對 的呼叫 AWS 服務 ，以及您使用 AWS SDK 進行的資源、對內部或外部 HTTP Web APIs呼叫，或 SQL 資料庫查詢。您也可以建立子區段，除錯或標註您應用程式中的程式碼區塊。子區段可包含其他子區段，因此記錄內部函數呼叫中繼資料的自訂子區段可包含其他自訂子區段及下游呼叫的子區段。

子區段會從呼叫它的服務觀點記錄下游呼叫。X-Ray 使用子區段來識別未傳送區段的下游服務，並在服務圖表上為其建立項目。

子區段可內嵌在完整區段文件中，或是分別傳送。將子區段分別傳送，來非同步追蹤長時間執行請求的下游呼叫，或是避免超過區段文件大小上限。

Example 具有內嵌子區段的區段

獨立子區段具備 subsegment 的 type，以及可識別父區段的 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
        }
      }
    }
  ]
```

```
}
```

對於長時間執行的請求，您可以傳送進行中區段以通知 X-Ray 已收到請求，然後單獨傳送子區段以追蹤它們，然後再完成原始請求。

Example 進行中的區段

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

Example 獨立子區段

獨立子區段具備 subsegment 的 type、trace_id 及 parent_id，可識別父區段。

```
{
  "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
      }
  }
}
```

當請求完成時，透過使用 end_time 重新傳送它來關閉區段。完整區段會覆寫正在進行中的區段。

您也可以為觸發非同步工作流程的已完成請求分別傳送子區段。例如，web API 可能會在開始使用者請求的工作前立即傳回 OK 200 回應。您可以在傳送回應後立即將完整區段傳送至 X-Ray，接著再將子區段傳送至稍後完成的工作。與區段相同，您也可以傳送子區段片段來記錄子區段已啟動，然後在完成下游呼叫時使用完整的子區段覆寫它。

以下欄位為子區段的必要項目或條件式必要項目。

> **Note**
>
> 除非另有說明，否則值必須為字串 (最多 250 個字元)。

必要的子區段欄位

- `id` – 子區段的 64 位元識別符，在相同追蹤中的區段中是唯一的，以 16 十六進位數字表示。
- `name` – 子區段的邏輯名稱。針對下游呼叫，請在呼叫資源或服務後命名子區段。針對自訂子區段，請在其檢測的程式碼 (例如函數名稱) 之後命名子區段。
- `start_time` – 子區段建立時間的數字，以 epoch 時間的浮點秒為單位，精確到毫秒。例如 `1480615200.010` 或 `1.480615200010E9`。
- `end_time` – 子區段關閉時間的數字。例如 `1480615200.090` 或 `1.480615200090E9`。指定 `end_time` 或 `in_progress`。
- `in_progress` – 設定為 `true`而非指定 的布林值`end_time`，以記錄子區段已啟動，但尚未完成。針對每個下游請求，請只傳送一個完成子區段，以及一或零個進行中子區段。
- `trace_id` – 子區段父區段的追蹤 ID。僅在分別傳送子區段時才為必要項目。

  X-Ray 追蹤 ID 格式

  X-Ray `trace_id`由以連字號分隔的三個數字組成。例如：`1-58406520-a006649127e371903a2de979`。其中包含：

  - 版本編號，即 1。
  - 使用 8 個十六進位數字的 Unix epoch 時間原始請求的時間。

    例如，10：00AM 2016 年 12 月 1 日 PST 以 epoch `58406520` 為單位是`1480615200`秒或十六進位數字。
  - 追蹤的全域唯一 96 位元識別符，以 24 十六進位數字表示。

> **Note**
>
> X-Ray 現在支援使用 OpenTelemetry 建立IDs，以及符合 W3C 追蹤內容規格的任何其他架構。W3C 追蹤 ID 在傳送至 X-Ray 時，必須以 X-Ray 追蹤 ID 格式格式化。例如，W3C 追蹤 ID 的格式`4efaaf4d1e8720b39541901950019ee5`應該如同傳送至 X-Ray `1-4efaaf4d-1e8720b39541901950019ee5`時一樣。X-Ray IDs 包含以 Unix epoch 時間表示的原始請求時間戳記，但在以 X-Ray 格式傳送 W3C 追蹤 IDs 時不需要。

- `parent_id` – 子區段父區段的區段 ID。僅在分別傳送子區段時才為必要項目。在巢狀子區段的情況下，子區段可以有一個區段或子區段作為其父項。
- `type` – `subsegment`。 只有在單獨傳送子區段時才需要。

以下欄位是子區段的選擇性欄位。

選擇性子區段欄位

- `namespace` – `aws`用於 AWS 開發套件呼叫； `remote` 用於其他下游呼叫。
- `http` – `http` 物件，其中包含傳出 HTTP 呼叫的相關資訊。
- `aws` – `aws` 物件，其中包含您應用程式呼叫的下游 AWS 資源相關資訊。
- `error`、`fault`、 `throttle`和 `cause` – 錯誤???欄位，指出發生錯誤，並包含導致錯誤的例外狀況相關資訊。
- `annotations` – 具有索引鍵/值對的`annotations`物件，您希望 X-Ray 為搜尋編製索引。
- `metadata` – `metadata` 物件，其中包含您要存放在區段的任何其他資料。
- `subsegments` – `subsegment` 物件陣列。
- `precursor_ids` – 子區段 IDs陣列，可識別具有在此子區段之前完成之相同父系的子區段。

HTTP 請求資料

使用 HTTP 區塊來記錄您應用程式所處理的 HTTP 請求相關詳細資訊 (位於區段中)，或是您應用程式對下游 HTTP API 所發出請求的相關詳細資訊 (位於子區段中)。此物件中大部分的欄位都會映射到 HTTP 請求和回應中找到的資訊。

`http`

所有欄位都是選擇性的。

- request – 請求的相關資訊。
  - method – 請求方法。例如：GET。
  - url – 請求的完整 URL，從請求的通訊協定、主機名稱和路徑編譯。
  - user_agent – 來自請求者用戶端的使用者代理程式字串。
  - client_ip – 申請者的 IP 地址。可從 IP 封包的 Source Address 擷取，或是針對轉送的請求，則從 X-Forwarded-For 標頭擷取。
  - x_forwarded_for –（僅限區段）布林值，指出 client_ip是從 X-Forwarded-For標頭讀取，而且不可靠，因為它可能是偽造的。
  - traced –（僅限子區段）布林值，表示下游呼叫是對另一個追蹤服務。如果此欄位設定為 true，X-Ray 會將追蹤視為中斷，直到下游服務上傳的區段parent_id與包含此區塊之子區段id的 相符。
- response – 回應的相關資訊。
  - status – 整數，指出回應的 HTTP 狀態。
  - content_length – 整數，表示回應內文的長度，以位元組為單位。

當您檢測對下游 Web api 的呼叫時，請記錄包含 HTTP 請求和回應相關資訊的子區段。X-Ray 使用 子區段來產生遠端 API 的推斷區段。

Example 在 Amazon EC2 上執行之應用程式提供的 HTTP 呼叫區段

```
{
  "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 呼叫的子區段

```
{
  "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 呼叫的推斷區段

```
{
  "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
}
```

註釋

區段和子區段可以包含annotations物件，其中包含一或多個欄位，X-Ray 索引可與篩選條件表達式搭配使用。欄位可以有字串、數字或布林值（無物件或陣列）。X-Ray 為每個追蹤編製最多 50 個註釋的索引。

Example HTTP 呼叫區段與標註

```
{
  "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
    }
  }
```

鍵必須為英數字元，才能使用篩選條件。允許使用底線。不允許使用其他符號和空格。

中繼資料

區段和子區段可以包含metadata物件，其中包含一或多個具有任何類型值的欄位，包括物件和陣列。只要區段文件不超過大小上限 (64 kB)，X-Ray 不會為中繼資料編製索引，且值可以是任何大小。您可以在 BatchGetTraces API 傳回的完整區段文件中檢視中繼資料。以 開頭的欄位金鑰 (debug在下列範例中） AWS. 會保留供 提供的 SDKs AWS和用戶端使用。

Example 使用中繼資料的自訂子區段

```
{
  "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 資源資料

針對區段，aws 物件包含您應用程式於其上執行的資源相關資訊。可將多個欄位套用至單一資源。例如，在 Elastic Beanstalk 上的多容器 Docker 環境中執行的應用程式，可能具有有關 Amazon EC2 執行個體、在執行個體上執行的 Amazon ECS 容器，以及 Elastic Beanstalk 環境本身的資訊。

aws (區段)

所有欄位都是選擇性的。

- `account_id` – 如果您的應用程式將客群傳送到不同的 AWS 帳戶，請記錄執行您應用程式的 帳戶 ID。
- `cloudwatch_logs` – 描述單一 CloudWatch 日誌群組的物件陣列。
  - `log_group` – CloudWatch Log Group 名稱。
  - `arn` – CloudWatch Log Group ARN。
- `ec2` – Amazon EC2 執行個體的相關資訊。
  - `instance_id` – EC2 執行個體的執行個體 ID。
  - `instance_size` – EC2 執行個體的類型。
  - `ami_id` – Amazon Machine Image ID。
  - `availability_zone` – 執行個體執行所在的可用區域。
- `ecs` – Amazon ECS 容器的相關資訊。
  - `container` – 容器的主機名稱。
  - `container_id` – 容器的完整容器 ID。

  - container_arn – 容器執行個體的 ARN。
- eks – Amazon EKS 叢集的相關資訊。
  - pod – EKS Pod 的主機名稱。
  - cluster_name – EKS 叢集名稱。
  - container_id – 容器的完整容器 ID。
- elastic_beanstalk – Elastic Beanstalk 環境的相關資訊。您可以在最新 Elastic Beanstalk 平台上名為 /var/elasticbeanstalk/xray/environment.conf的檔案中找到此資訊。
  - environment_name - 環境名稱。
  - version_label – 目前部署到提供請求之執行個體的應用程式版本名稱。
  - deployment_id – 數字，指出上次成功部署到提供請求之執行個體的 ID。
- xray – 有關所用檢測類型和版本的中繼資料。
  - auto_instrumentation – 布林值，指出是否使用自動檢測 （例如 Java 代理程式）。
  - sdk_version – 正在使用的 SDK 或代理程式版本。
  - sdk – SDK 的類型。

Example AWS 具有外掛程式的 區塊

```
"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"
   }
}
```

針對子區段，記錄應用程式存取之 AWS 服務 和資源的相關資訊。X-Ray 會使用此資訊來建立推斷區段，代表服務映射中的下游服務。

aws (子區段)

所有欄位都是選擇性的。

- `operation` – 針對 AWS 服務 或 資源叫用的 API 動作名稱。
- `account_id` – 如果您的應用程式存取不同帳戶中的資源，或將客群傳送至不同帳戶，請記錄擁有應用程式存取之 AWS 資源的帳戶 ID。
- `region` – 如果資源所在的區域與您的應用程式不同，請記錄區域。例如：`us-west-2`。
- `request_id` – 請求的唯一識別符。
- `queue_url` – 對於 Amazon SQS 佇列上的操作，則為佇列的 URL。
- `table_name` – 對於 DynamoDB 資料表上的操作，則為資料表的名稱。

Example 呼叫 DynamoDB 以儲存項目的子區段

```
{
  "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",
  }
}
```

錯誤和例外狀況

當發生錯誤時，您可以記錄錯誤及其產生異常的相關詳細資訊。在您應用程式將錯誤傳回使用者時於區段中記錄錯誤，或是在下游呼叫傳回錯誤時於子區段中記錄錯誤。

錯誤類型

將一或多個以下欄位設為 `true` 來指出發生錯誤。若錯誤發生複合，則可套用多個類型。例如，來自下游呼叫的 `429 Too Many Requests` 錯誤可能會造成您的應用程式傳回 `500 Internal Server Error`；在這種情況下，即會套用三種類型。

- `error` – 布林值表示發生用戶端錯誤 （回應狀態碼為 4XX 用戶端錯誤）。
- `throttle` – 布林值，指出請求已調節 （回應狀態碼為 429 太多請求)。
- `fault` – 布林值表示發生伺服器錯誤 （回應狀態碼為 5XX 伺服器錯誤）。

透過在區段或子區段中包含一個原因 (cause) 物件來指出錯誤的原因。

`cause`

原因可以是 16 字元的異常 ID，或是具備以下欄位的物件：

- `working_directory` – 發生例外狀況時工作目錄的完整路徑。
- `paths` – 發生例外狀況時使用的程式庫或模組的路徑陣列。
- `exceptions` – 例外狀況物件的陣列。

在一或多個異常 (exception) 物件中包含錯誤的詳細資訊。

`exception`

所有欄位都是選擇性的。

- `id` – 例外狀況的 64 位元識別符，在相同追蹤中的區段中是唯一的，以 16 個十六進位數字表示。
- `message` – 例外狀況訊息。
- `type` – 例外狀況類型。
- `remote` – 布林值，指出例外是由下游服務傳回的錯誤所造成。
- `truncated` – 整數，指出從 省略的堆疊影格數目`stack`。
- `skipped` – 整數，指出在此例外狀況及其子項之間略過的例外狀況數目，也就是它導致的例外狀況。

- `cause` – 例外狀況的父系例外狀況 ID，也就是造成此例外狀況的例外狀況。
- `stack` – stackFrame 物件陣列。

若可用的話，在 stackFrame 物件中記錄呼叫堆疊的相關資訊。

`stackFrame`

所有欄位都是選擇性的。

- `path` – 檔案的相對路徑。
- `line` – 檔案中的行。
- `label` – 函數或方法名稱。

SQL 查詢

您可以建立為您應用程式對 SQL 資料庫發出的查詢建立子區段。

`sql`

所有欄位都是選擇性的。

- `connection_string` – 對於不使用 URL 連線字串的 SQL Server 或其他資料庫連線，請記錄連線字串，但不包括密碼。
- `url` – 對於使用 URL 連線字串的資料庫連線，請記錄 URL，不包括密碼。
- `sanitized_query` – 資料庫查詢，其中任何使用者提供的值都會移除或由預留位置取代。
- `database_type` – 資料庫引擎的名稱。
- `database_version` – 資料庫引擎的版本編號。
- `driver_version` – 您的應用程式使用的資料庫引擎驅動程式名稱和版本編號。
- `user` – 資料庫使用者名稱。
- `preparation` – `call` 如果查詢使用 `PreparedCall`；`statement`如果查詢使用 `PreparedStatement`。

Example 使用 SQL 查詢的子區段

```
{
  "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 會接收來自 服務的資料做為區段。然後，X-Ray 會將具有常見請求的客群分組為追蹤。X-Ray 會處理追蹤以產生服務圖表，以提供應用程式的視覺化呈現。

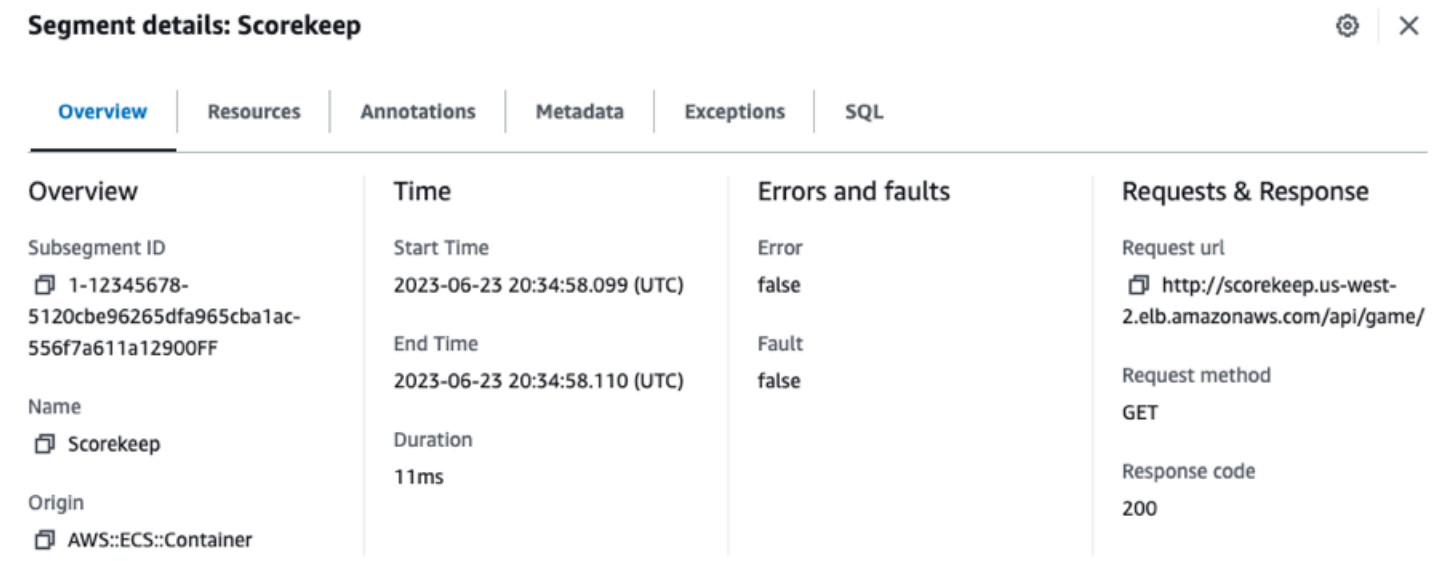
概念

- 客群
- 子區段
- 服務圖表
- 追蹤
- 抽樣
- 追蹤標頭
- 篩選條件表達式
- 群組
- 標註和中繼資料
- 錯誤、故障和例外狀況

客群

執行應用程式邏輯的運算資源會以區段形式傳送與其工作相關的資料。區段中提供資源的名稱、請求的詳細資訊，以及完成的工作詳細資訊。例如，當 HTTP 請求到達您的應用程式時，它可以記錄以下相關資料：

- 主機 – 主機名稱、別名或 IP 地址
- 請求 – 方法、用戶端地址、路徑、使用者代理程式
- 回應 – 狀態、內容
- 完成的工作 – 開始和結束時間、子區段
- 發生的問題 – 錯誤、故障和例外狀況，包括自動擷取例外狀況堆疊。



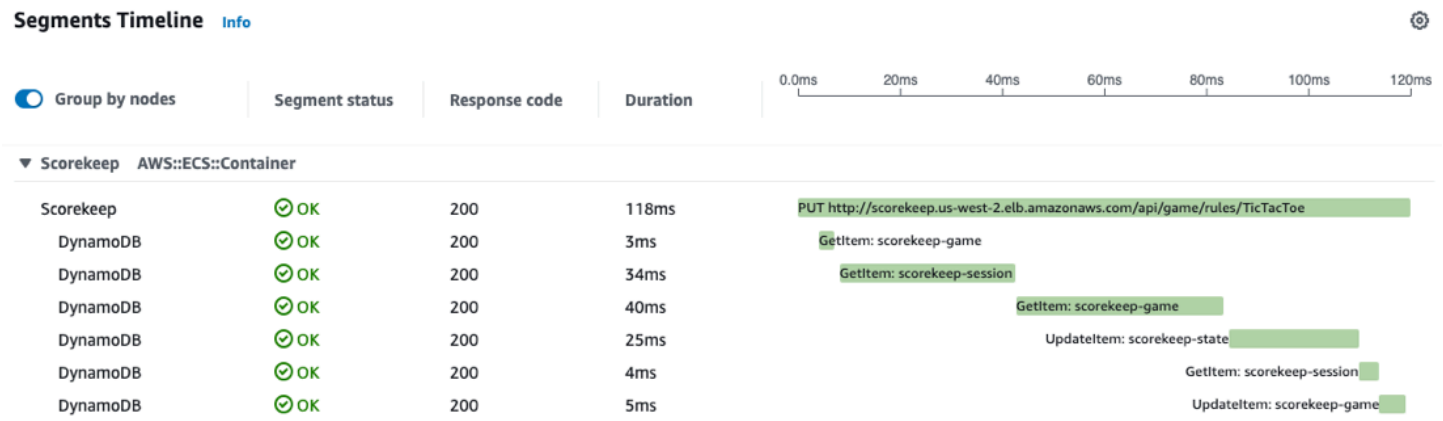
X-Ray SDK 會從請求和回應標頭、應用程式中的程式碼，以及執行 AWS 資源的中繼資料中收集資訊。您可以透過修改應用程式組態或程式碼來檢測傳入請求、下游請求和 AWS SDK 用戶端，來選擇要收集的資料。

> **轉送的請求**
>
> 如果負載平衡器或其他中介裝置轉送請求到您的應用程式，X-Ray 會從請求中的 `X-Forwarded-For`標頭取得用戶端 IP，而不是從 IP 封包中的來源 IP 取得用戶端 IP。為轉送請求記錄的用戶端 IP 可能是偽造的，因此不應受信任。

您可以使用 X-Ray SDK 記錄其他資訊，例如註釋和中繼資料。如需結構詳細資訊與區段和子區段中所記錄的資訊，請參閱AWS X-Ray 區段文件。區段文件的大小上限為 64 kB。

子區段

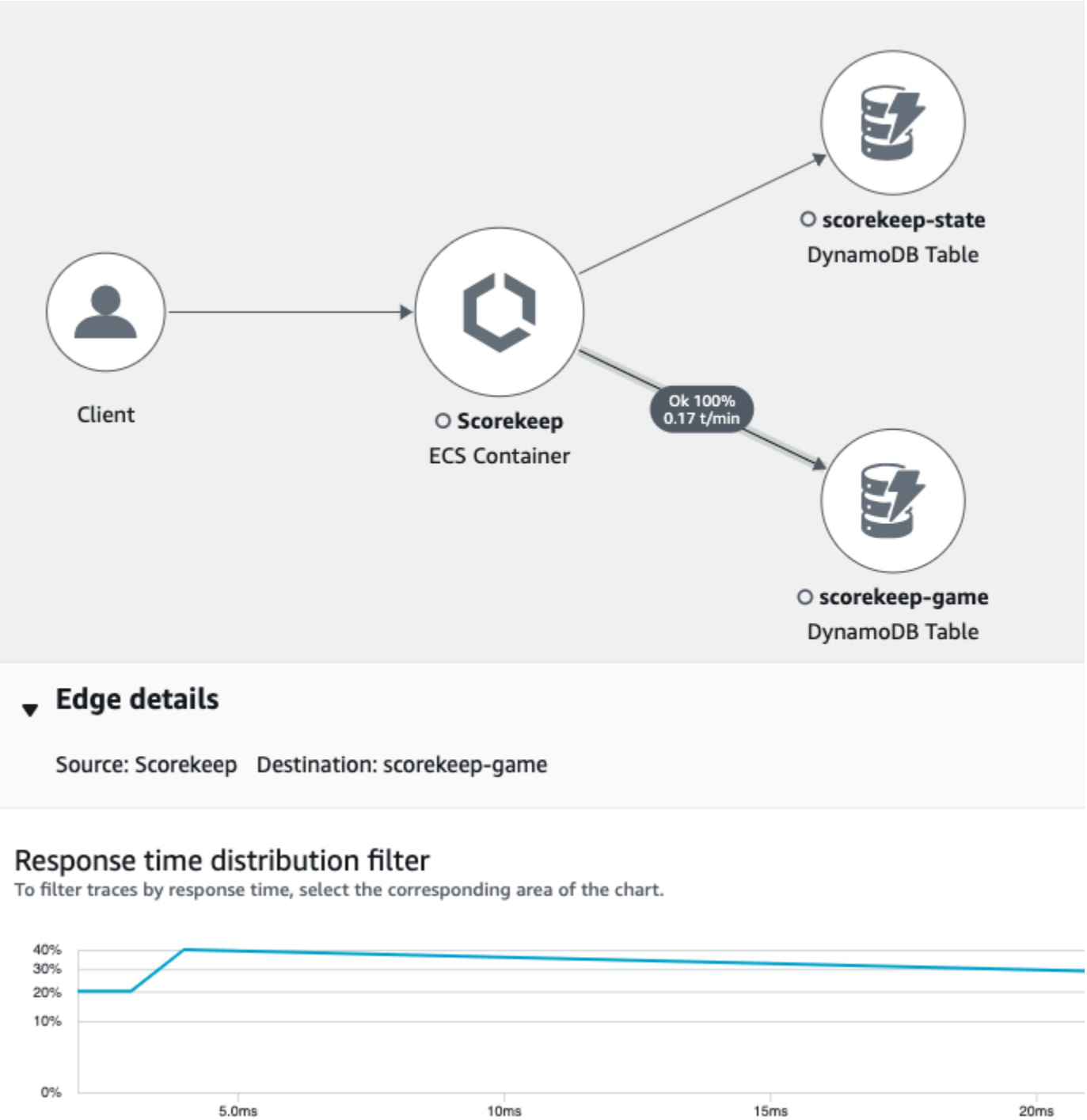
區段可以將完成工作的資料細分為子區段。子區段可提供更精確的計時資訊，以及應用程式為滿足原始請求所做的下游呼叫詳細資訊。子區段可以包含有關呼叫 AWS 服務、外部 HTTP API 或 SQL 資料庫的其他詳細資訊。您甚至可以定義任意子區段來檢測應用程式的特定函數或程式碼行。



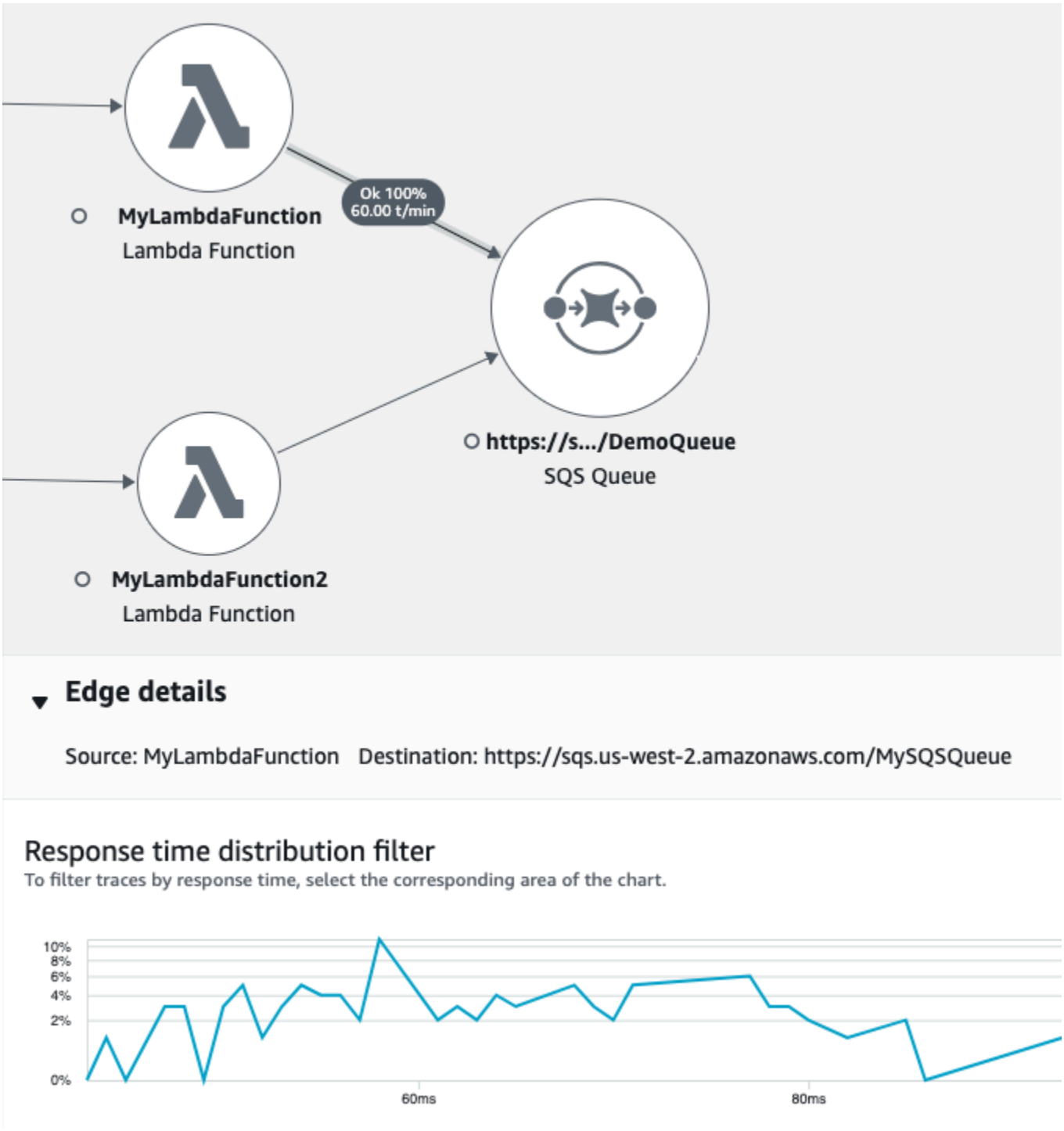
對於不傳送自己的區段的服務，例如 Amazon DynamoDB，X-Ray 會使用子區段在追蹤映射上產生推斷區段和下游節點。這可讓您查看所有下游相依性，即使這些相依性不支援追蹤，或屬於外部相依性亦同。

子區段會以用戶端形式來代表您應用程式的下游呼叫檢視。如果下游服務也經過檢測，其傳送的區段則會取代從上游用戶端子區段產生的推斷區段。服務圖表的節點一律使用來自服務區段的資訊；兩個節點之間的邊緣則會使用上游服務的子區段。

例如，當您使用經檢測的 AWS SDK 用戶端呼叫 DynamoDB 時，X-Ray SDK 會記錄該呼叫的子區段。DynamoDB 不會傳送區段，因此追蹤中的推斷區段、服務圖表上的 DynamoDB 節點，以及您的服務與 DynamoDB 之間的邊緣都包含子區段中的資訊。



當您使用經檢測的應用程式呼叫其他經檢測的服務時，下游服務會傳送自己的區段以記錄上游服務在子區段中記錄的相同呼叫檢視。在服務圖表中，兩種服務的節點都會包含來自這些服務區段的計時和錯誤資訊，而之間的邊緣則包含來自上游服務子區段的資訊。



這兩種檢視都很實用，因為下游服務會精確記錄請求的工作開始和結束時間，而上游服務會記錄請求的往返延遲，包括請求在這兩項服務之間移動所花的時間。

服務圖表

X-Ray 會使用應用程式傳送的資料來產生服務圖表。將資料傳送至 X-Ray 的每個 AWS 資源都會在圖形中顯示為服務。邊緣可連線至各種服務，這些服務則協力為請求提供服務。邊緣會將用戶端連線至您的應用程式，並將您的應用程式連線至其所用的下游服務和資源。

> **服務名稱**
>
> 區段的 name應與產生區段之服務的網域名稱或邏輯名稱相符。不過，這不會強制執行。具有許可的任何應用程式[PutTraceSegments]都可以以任何名稱傳送客群。

服務圖表是一種 JSON 文件，其中包含構成您應用程式的服務和資源相關資訊。X-Ray 主控台使用服務圖表來產生視覺化或服務映射。

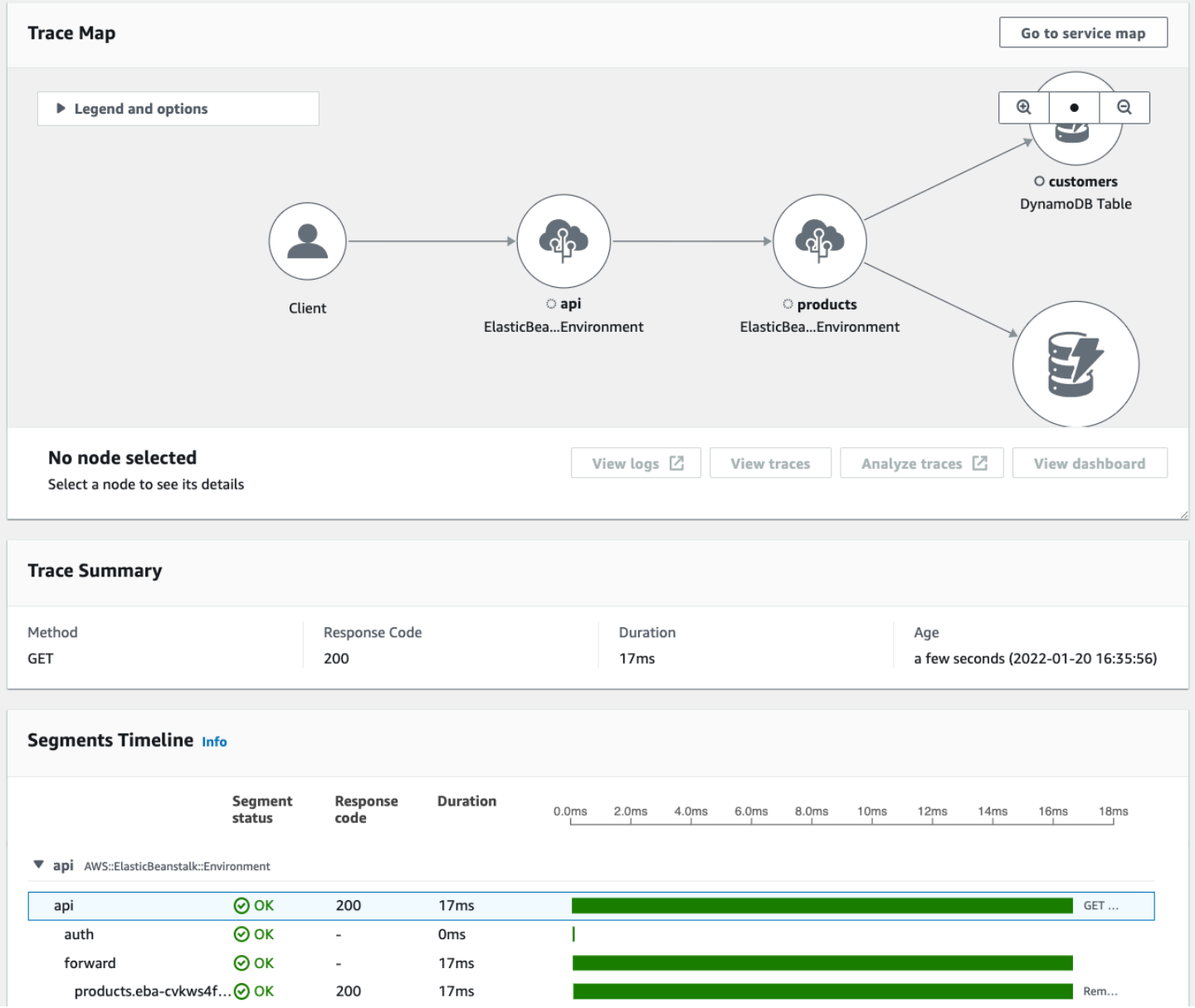
對於分散式應用程式，X-Ray 會將處理具有相同追蹤 ID 之請求的所有 服務節點合併為單一服務圖表。請求命中的第一個服務會新增追蹤標頭，而系統會在前端和其呼叫的服務之間傳播此標頭。

例如，Scorekeep 會執行呼叫微服務的 Web API (AWS Lambda 函數)，以使用 Node.js 程式庫來產生隨機名稱。適用於 Java 的 X-Ray 開發套件會產生追蹤 ID，並將其包含在對 Lambda 的呼叫中。Lambda 會傳送追蹤資料，並將追蹤 ID 傳遞給函數。適用於 Node.js 的 X-Ray 開發套件也會使用追蹤 ID 來傳送資料。因此，API、Lambda 服務和 Lambda 函數的節點都會在追蹤映射上顯示為獨立但已連線的節點。

服務圖表資料會保留 30 天。

追蹤

追蹤 ID 可追蹤透過應用程式的請求路徑。追蹤會收集由單一請求所產生的所有區段。該請求通常是 HTTP GET 或 POST 請求，它會通過負載平衡器、命中您的應用程式程式碼，以及產生對其他服務 AWS 或外部 Web APIs下游呼叫。第一個與 HTTP 請求互動的支援服務會為請求新增追蹤 ID，並向下游傳播以追蹤延遲、處理和其他請求資料。



如需 X-Ray 追蹤計費方式的資訊，請參閱AWS X-Ray 定價。追蹤資料會保留 30 天。

抽樣

為了確保有效率的追蹤並提供應用程式提供的請求代表性範例，X-Ray SDK 會套用取樣演算法來判斷要追蹤哪些請求。根據預設，X-Ray 開發套件每秒記錄第一個請求，以及任何其他請求的 5%。

為了避免開始使用時產生的服務費用，預設的抽樣費率都很保守。您可以設定 X-Ray 來修改預設抽樣規則，並設定其他規則，根據服務或請求的屬性套用抽樣。

例如，您可能想要停用取樣，並追蹤所有修改狀態或處理使用者或交易的呼叫請求。若是大量唯讀呼叫 (例如背景輪詢、運作狀態檢查或連線維護)，您可以用低費率抽樣但仍獲得足夠的資料，以查看發生的任何問題。

如需詳細資訊，請參閱設定 取樣規則。

追蹤標頭

所有請求都會追蹤，並到可設定的最低限度為止。到達該最低限度之後，就會追蹤某個百分比的請求，以避免不必要的成本。取樣決策和追蹤 ID 會新增至名為 的追蹤標頭中的 HTTP 請求`X-Amzn-Trace-Id`。請求命中的第一個 X-Ray-integrated服務會新增追蹤標頭，由 X-Ray SDK 讀取並包含在回應中。

Example 含根追蹤 ID 和抽樣決策的追蹤標頭

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

> **追蹤標頭的安全性**
>
> 追蹤標頭可以源自 X-Ray SDK、 AWS 服務或用戶端請求。您的應用程式可以從傳入的請求移除 `X-Amzn-Trace-Id`，以避免使用者將追蹤 ID 或抽樣決策新增到請求時發生問題。

如果請求是從經檢測的應用程式產生，則追蹤標頭也可以包含父區段 ID。例如，如果您的應用程式使用經檢測的 HTTP 用戶端呼叫下游 HTTP Web API，X-Ray SDK 會將原始請求的區段 ID 新增至下游請求的追蹤標頭。為下游請求提供服務的經檢測應用程式，可以記錄父區段 ID 以連線這兩個請求。

Example 含根追蹤 ID、父區段 ID 和抽樣決策的追蹤標頭

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

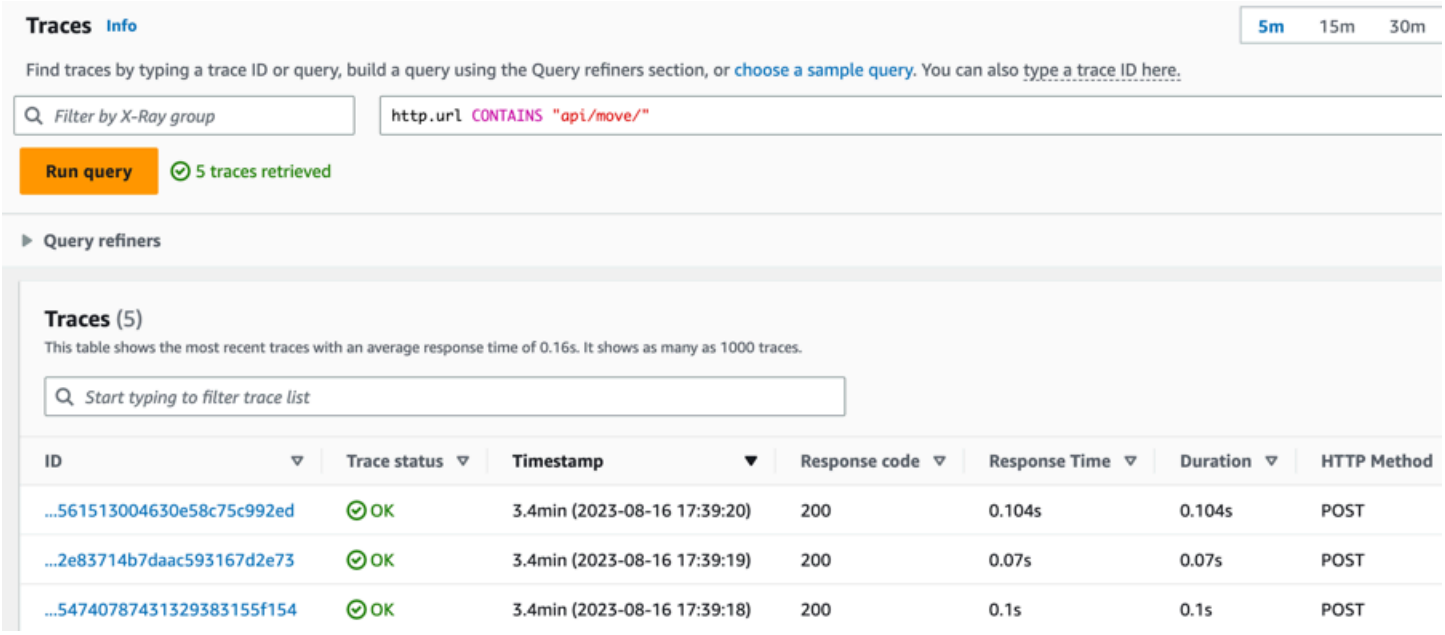
`Lineage` 可能會由 Lambda 和其他 附加到追蹤標頭 AWS 服務 ，作為其處理機制的一部分，不應直接使用。

Example 使用 Lineage 追蹤標頭

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

篩選條件表達式

即便使用抽樣，複雜的應用程式仍會產生大量資料。 AWS X-Ray 主控台提供easy-to-navigate的服務圖表檢視。它會顯示運作狀況和效能資訊，以協助您識別問題並找出最佳化應用程式的機會。若要進階追蹤，您可以向下切入以追蹤個別請求，或使用篩選條件表達式，以尋找與特定路徑或使用者相關的追蹤。



群組

X-Ray 擴充篩選條件表達式也支援 群組功能。使用篩選條件表達式，即可以定義條件，以接受追蹤到群組。

您可以依名稱或 Amazon Resource Name (ARN) 呼叫 群組，以產生自己的服務圖表、追蹤摘要和 Amazon CloudWatch 指標。建立群組後，系統會針對群組的篩選條件表達式檢查傳入的追蹤，因為這些追蹤存放在 X-Ray 服務中。符合每個條件的追蹤數量指標每分鐘都會發佈至 CloudWatch。

更新群組的篩選條件表達式不會變更已記錄的資料。更新僅適用於後續追蹤。這會導致圖表合併新舊表達式。若要避免這種情況，請刪除目前群組並建立新的群組。

> **Note**
>
> 群組的計費方式是根據符合篩選條件表達式的擷取追蹤。如需詳細資訊，請參閱 AWS X-Ray 定價。

如需 群組的詳細資訊，請參閱 設定 群組。

標註和中繼資料

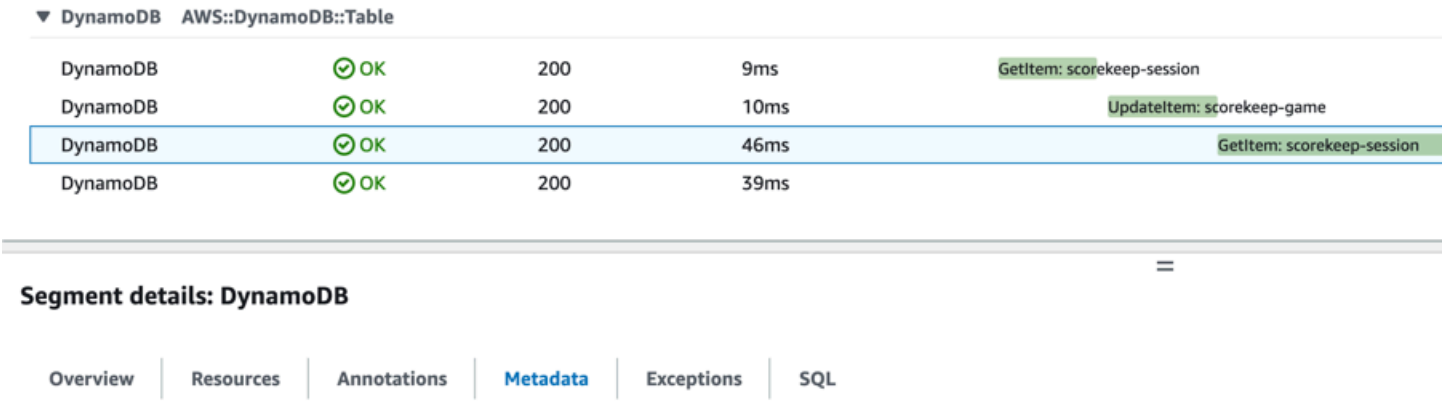
當您檢測應用程式時，X-Ray 開發套件會記錄傳入和傳出請求、使用 AWS 的資源和應用程式本身的相關資訊。您可以藉由註釋和中繼資料形式，將其他資訊新增至區段文件。註釋和中繼資料會在追蹤層級彙總，並且可以新增至任何區段或子區段。

註釋是簡單的鍵/值對，其會建立索引以與篩選條件表達式搭配使用。使用標記記錄您想要用來在主控台將追蹤分組的資料，或是在呼叫 `GetTraceSummaries` API 時使用標記。

X-Ray 為每個追蹤編製最多 50 個註釋的索引。

中繼資料為含有任何類型值的鍵/值對，包括物件和清單，但不會建立索引。您可以使用中繼資料，來記錄想要存放於追蹤但不需用於搜尋追蹤的資料。

您可以在 CloudWatch 主控台的追蹤詳細資訊頁面中，檢視區段或子區段詳細資訊視窗中的註釋和中繼資料。



錯誤、故障和例外狀況

X-Ray 會追蹤應用程式程式碼中發生的錯誤，以及下游服務傳回的錯誤。錯誤分類如下。

- **`Error`** – 用戶端錯誤 (400 系列錯誤)
- **`Fault`** – 伺服器故障 (500 系列錯誤)
- **`Throttle`** – 調節錯誤 (429 個請求過多)

當您的應用程式提供經檢測的請求時發生例外狀況時，X-Ray 開發套件會記錄例外狀況的詳細資訊，包括可用的堆疊追蹤。您可以在 X-Ray 主控台的區段詳細資訊下檢視例外狀況。

中的安全性 AWS X-Ray

的雲端安全 AWS 是最高優先順序。身為 AWS 客戶，您可以受益於資料中心和網路架構，這些架構專為滿足最安全敏感組織的需求而建置。

安全是 AWS 與您之間的共同責任。共同責任模型 將此描述為雲端的安全和雲端內的安全：

- 雲端的安全性 – AWS 負責保護在 AWS 服務 中執行的基礎設施 AWS 雲端。 AWS 也為您提供可安全使用的服務。第三方稽核人員定期檢測及驗證安全的效率也是我們 AWS 合規計劃的一部分。若要了解適用於 X-Ray 的合規計劃，請參閱AWS 服務 合規計劃範圍中的 。
- 雲端的安全性 – 您的責任取決於您使用 AWS 服務 的 。您也必須對資料敏感度、組織要求，以及適用法律和法規等其他因素負責。

本文件將協助您了解如何在使用 X-Ray 時套用共同責任模型。下列主題說明如何設定 X-Ray 以符合您的安全與合規目標。您也將了解如何使用其他 AWS 服務 來協助您監控和保護 X-Ray 資源。

主題

- 中的資料保護 AWS X-Ray
- 的身分和存取管理 AWS X-Ray
- 的合規驗證 AWS X-Ray
- 中的彈性 AWS X-Ray
- 中的基礎設施安全 AWS X-Ray

中的資料保護 AWS X-Ray

AWS X-Ray 一律加密追蹤和靜態相關資料。當您需要稽核和停用加密金鑰以符合合規或內部要求時，您可以將 X-Ray 設定為使用 AWS Key Management Service (AWS KMS) 金鑰來加密資料。

X-Ray 提供 AWS 受管金鑰 名為 的 `aws/xray`。當您只想要稽核 AWS CloudTrail中的金鑰使用情況而不需要管理金鑰本身時，請使用此金鑰。當您需要管理金鑰的存取權或設定金鑰輪換時，您可以建立客戶受管金鑰。

當您變更加密設定時，X-Ray 會花費一些時間來產生和傳播資料金鑰。雖然會處理新的金鑰，但 X-Ray 可能會使用新設定和舊設定的組合來加密資料。當您變更加密設定時，並不會重新加密現有資料。

> **Note**
>
> AWS KMS 當 X-Ray 使用 KMS 金鑰來加密或解密追蹤資料時， 會收取費用。
>
> - 預設加密 – 免費。
> - AWS 受管金鑰 – 支付金鑰使用費。
> - 客戶受管金鑰 – 支付金鑰儲存和使用的費用。
>
> 如需詳細資訊，請參閱 AWS Key Management Service 定價。

> **Note**
>
> X-Ray 洞察通知會將事件傳送至 Amazon EventBridge，目前不支援客戶受管金鑰。如需詳細資訊，請參閱 Amazon EventBridge 中的資料保護。

您必須擁有客戶受管金鑰的使用者層級存取權，才能將 X-Ray 設定為使用該金鑰，然後檢視加密的追蹤。如需更多資訊，請參閱用於加密的使用者許可。

CloudWatch console

設定 X-Ray 使用 CloudWatch 主控台使用 KMS 金鑰進行加密

1. 登入 AWS Management Console ，並在 https://console.aws.amazon.com/cloudwatch/ 開啟 CloudWatch 主控台。
2. 在左側導覽窗格中選擇設定。
3. 在 X-Ray 追蹤區段的加密下選擇檢視設定。
4. 在加密組態區段中選擇編輯。
5. 選擇使用 KMS 金鑰。
6. 從下拉式選單中選擇金鑰：
   - aws/xray – 使用 AWS 受管金鑰。
   - 金鑰別名 – 在您的帳戶中使用客戶受管金鑰。
   - 手動輸入金鑰 ARN – 在不同帳戶中使用客戶受管金鑰。在顯示欄位中，輸入金鑰的完整 Amazon Resource Name (ARN)。

7. 選擇更新加密。

X-Ray console

設定 X-Ray 使用 X-Ray 主控台使用 KMS 金鑰進行加密

1. 開啟 X-Ray 主控台。
2. 選擇 Encryption (加密)。
3. 選擇使用 KMS 金鑰。
4. 從下拉式選單中選擇金鑰：
   - aws/xray – 使用 AWS 受管金鑰。
   - 金鑰別名 – 在您的帳戶中使用客戶受管金鑰。
   - 手動輸入金鑰 ARN – 在不同帳戶中使用客戶受管金鑰。在顯示欄位中，輸入金鑰的完整 Amazon Resource Name (ARN)。
5. 選擇套用。

> **Note**
>
> X-Ray 不支援非對稱 KMS 金鑰。

如果 X-Ray 無法存取您的加密金鑰，則會停止儲存資料。如果您的使用者無法存取 KMS 金鑰，或者您停用目前正在使用的金鑰，就可能會發生這種情況。發生這種情況時，X-Ray 會在導覽列中顯示通知。

若要使用 X-Ray API 設定加密設定，請參閱 使用 AWS X-Ray API 設定抽樣、分組和加密設定。

的身分和存取管理 AWS X-Ray

AWS Identity and Access Management (IAM) 是 AWS 服務 ，可協助管理員安全地控制對 AWS 資源的存取。IAM 管理員可控制誰可以進行身分驗證 （ 登入 ） 和授權 （ 具有許可 ） 來使用 X-Ray 資源。IAM 是您可以免費使用 AWS 服務 的 。

主題

- 目標對象

- 使用身分驗證
- 使用政策管理存取權
- AWS X-Ray 如何使用 IAM
- AWS X-Ray 身分型政策範例
- 對 AWS X-Ray 身分和存取進行故障診斷

目標對象

您使用 AWS Identity and Access Management (IAM) 的方式會有所不同，取決於您在 X-Ray 中執行的工作。

服務使用者 – 如果您使用 X-Ray 服務來執行任務，管理員會為您提供所需的登入資料和許可。當您使用更多 X-Ray 功能來執行工作時，您可能需要額外的許可。了解存取許可的管理方式可協助您向管理員請求正確的許可。如果您無法存取 X-Ray 中的功能，請參閱 對 AWS X-Ray 身分和存取進行故障診斷。

服務管理員 – 如果您在公司負責 X-Ray 資源，您可能擁有 X-Ray 的完整存取權。您的任務是判斷服務使用者應存取的 X-Ray 功能和資源。接著，您必須將請求提交給您的 IAM 管理員，來變更您服務使用者的許可。檢閱此頁面上的資訊，了解 IAM 的基本概念。若要進一步了解貴公司如何搭配 X-Ray 使用 IAM，請參閱 AWS X-Ray 如何使用 IAM。

IAM 管理員 – 如果您是 IAM 管理員，建議您了解撰寫政策以管理 X-Ray 存取的詳細資訊。若要檢視您可以在 IAM 中使用的 X-Ray 身分型政策範例，請參閱 AWS X-Ray 身分型政策範例。

使用身分驗證

身分驗證是您 AWS 使用身分憑證登入 的方式。您必須以 AWS 帳戶根使用者身分、IAM 使用者身分或擔任 IAM 角色來驗證 （ 登入 AWS)。

您可以使用透過身分來源提供的憑證，以聯合身分 AWS 身分身分身分身分登入 。 AWS IAM Identity Center (IAM Identity Center) 使用者、您公司的單一登入身分驗證，以及您的 Google 或 Facebook 登入資料，都是聯合身分的範例。您以聯合身分登入時，您的管理員先前已設定使用 IAM 角色的聯合身分。當您使用聯合 AWS 身分存取 時，您會間接擔任角色。

根據您身分的使用者類型，您可以登入 AWS Management Console 或 AWS 存取入口網站。如需登入的詳細資訊 AWS，請參閱AWS 登入 《 使用者指南》中的如何登入您的 AWS 帳戶 。

如果您以 AWS 程式設計方式存取 ， AWS 會提供軟體開發套件 (SDK) 和命令列界面 (CLI)，以使用您的登入資料以密碼編譯方式簽署您的請求。如果您不使用 AWS 工具，則必須自行簽署請求。如需

使用建議的方法自行簽署請求的詳細資訊，請參閱《IAM 使用者指南》中的適用於 API 請求的AWS Signature 第 4 版。

無論您使用何種身分驗證方法，您可能都需要提供額外的安全性資訊。例如， AWS 建議您使用多重驗證 (MFA) 來提高帳戶的安全性。如需更多資訊，請參閱《AWS IAM Identity Center 使用者指南》中的多重要素驗證和《IAM 使用者指南》中的 IAM 中的AWS 多重要素驗證。

AWS 帳戶 根使用者

當您建立 時 AWS 帳戶，您會從一個登入身分開始，該身分可完整存取帳戶中的所有 AWS 服務 和 資源。此身分稱為 AWS 帳戶 Theroot 使用者，可透過使用您用來建立帳戶的電子郵件地址和密碼登入來存取。強烈建議您不要以根使用者處理日常任務。保護您的根使用者憑證，並將其用來執行只能由根使用者執行的任務。如需這些任務的完整清單，了解需以根使用者登入的任務，請參閱《IAM 使用者指南》中的需要根使用者憑證的任務。

IAM 使用者和群組

IAM 使用者是 中的身分 AWS 帳戶 ，具有單一人員或應用程式的特定許可。建議您盡可能依賴臨時憑證，而不是擁有建立長期憑證 (例如密碼和存取金鑰) 的 IAM 使用者。但是如果特定使用案例需要擁有長期憑證的 IAM 使用者，建議您輪換存取金鑰。如需更多資訊，請參閱 IAM 使用者指南中的為需要長期憑證的使用案例定期輪換存取金鑰。

IAM 群組是一種指定 IAM 使用者集合的身分。您無法以群組身分簽署。您可以使用群組來一次為多名使用者指定許可。群組可讓管理大量使用者許可的程序變得更為容易。例如，您可以擁有一個名為 IAMAdmins 的群組，並給予該群組管理 IAM 資源的許可。

使用者與角色不同。使用者只會與單一人員或應用程式建立關聯，但角色的目的是在由任何需要它的人員取得。使用者擁有永久的長期憑證，但角色僅提供臨時憑證。如需更多資訊，請參閱《IAM 使用者指南》中的 IAM 使用者的使用案例。

IAM 角色

IAM 角色是 中具有特定許可 AWS 帳戶 的身分。它類似 IAM 使用者，但不與特定的人員相關聯。若要暫時在 中擔任 IAM 角色 AWS Management Console，您可以從使用者切換至 IAM 角色 （主控台）。您可以透過呼叫 AWS CLI 或 AWS API 操作或使用自訂 URL 來擔任角色。如需使用角色的方法詳細資訊，請參閱《IAM 使用者指南》中的擔任角色的方法。

使用臨時憑證的 IAM 角色在下列情況中非常有用：

- 聯合身分使用者存取 — 如需向聯合身分指派許可，請建立角色，並為角色定義許可。當聯合身分進行身分驗證時，該身分會與角色建立關聯，並獲授予由角色定義的許可。如需有關聯合角色的相關資

訊，請參閱《IAM 使用者指南》中的為第三方身分提供者 (聯合) 建立角色。如果您使用 IAM Identity Center，則需要設定許可集。為控制身分驗證後可以存取的內容，IAM Identity Center 將許可集與 IAM 中的角色相關聯。如需有關許可集的資訊，請參閱 AWS IAM Identity Center 使用者指南中的許可集。

- 暫時 IAM 使用者許可 – IAM 使用者或角色可以擔任 IAM 角色來暫時針對特定任務採用不同的許可。
- 跨帳戶存取權：您可以使用 IAM 角色，允許不同帳戶中的某人 (信任的主體) 存取您帳戶的資源。角色是授予跨帳戶存取權的主要方式。不過，對於某些 AWS 服務，您可以將政策直接連接到資源（而不是使用角色做為代理）。如需了解使用角色和資源型政策進行跨帳戶存取之間的差異，請參閱《IAM 使用者指南》中的 IAM 中的跨帳戶資源存取。
- 跨服務存取 – 有些 AWS 服務 使用其他 中的功能 AWS 服務。例如，當您在服務中進行呼叫時，該服務通常會在 Amazon EC2 中執行應用程式或將物件儲存在 Amazon Simple Storage Service (Amazon S3) 中。服務可能會使用呼叫主體的許可、使用服務角色或使用服務連結角色來執行此作業。
  - 轉送存取工作階段 (FAS) – 當您使用 IAM 使用者或角色在其中執行動作時 AWS，您被視為委託人。使用某些服務時，您可能會執行某個動作，進而在不同服務中啟動另一個動作。FAS 使用呼叫 的委託人許可 AWS 服務，結合 AWS 服務 請求向下游服務提出請求。只有當服務收到需要與其他 AWS 服務 或 資源互動才能完成的請求時，才會提出 FAS 請求。在此情況下，您必須具有執行這兩個動作的許可。如需提出 FAS 請求時的政策詳細資訊，請參閱《轉發存取工作階段》。
  - 服務角色 – 服務角色是服務擔任的 IAM 角色，可代表您執行動作。IAM 管理員可以從 IAM 內建立、修改和刪除服務角色。如需詳細資訊，請參閱《IAM 使用者指南》中的建立角色以委派許可權給 AWS 服務。
  - 服務連結角色 – 服務連結角色是一種連結至 的服務角色類型 AWS 服務。服務可以擔任代表您執行動作的角色。服務連結角色會出現在您的 中 AWS 帳戶 ，並由服務擁有。IAM 管理員可以檢視，但不能編輯服務連結角色的許可。
- 在 Amazon EC2 上執行的應用程式 – 您可以使用 IAM 角色來管理在 EC2 執行個體上執行之應用程式的臨時登入資料，以及提出 AWS CLI 或 AWS API 請求。這是在 EC2 執行個體內儲存存取金鑰的較好方式。若要將 AWS 角色指派給 EC2 執行個體，並將其提供給其所有應用程式，您可以建立連接至執行個體的執行個體描述檔。執行個體設定檔包含該角色，並且可讓 EC2 執行個體上執行的程式取得臨時憑證。如需詳細資訊，請參閱《IAM 使用者指南》中的使用 IAM 角色來授予許可權給 Amazon EC2 執行個體上執行的應用程式。

使用政策管理存取權

您可以透過建立政策並將其連接到身分或資源 AWS 來控制 AWS 中的存取。政策是 中的物件，當與身分或資源建立關聯時， AWS 會定義其許可。當委託人（使用者、根使用者或角色工作階段）發出

請求時， 會 AWS 評估這些政策。政策中的許可決定是否允許或拒絕請求。大多數政策會以 JSON 文件 AWS 的形式存放在 中。如需 JSON 政策文件結構和內容的詳細資訊，請參閱 IAM 使用者指南中的 JSON 政策概觀。

管理員可以使用 AWS JSON 政策來指定誰可以存取內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

預設情況下，使用者和角色沒有許可。若要授予使用者對其所需資源執行動作的許可，IAM 管理員可以建立 IAM 政策。然後，管理員可以將 IAM 政策新增至角色，使用者便能擔任這些角色。

IAM 政策定義該動作的許可，無論您使用何種方法來執行操作。例如，假設您有一個允許 `iam:GetRole` 動作的政策。具有該政策的使用者可以從 AWS Management Console AWS CLI、 或 AWS API 取得角色資訊。

身分型政策

身分型政策是可以附加到身分 (例如 IAM 使用者、使用者群組或角色) 的 JSON 許可政策文件。這些政策可控制身分在何種條件下能對哪些資源執行哪些動作。如需了解如何建立身分型政策，請參閱《IAM 使用者指南》中的透過客戶管理政策定義自訂 IAM 許可。

身分型政策可進一步分類成內嵌政策或受管政策。內嵌政策會直接內嵌到單一使用者、群組或角色。受管政策是獨立的政策，您可以連接到 中的多個使用者、群組和角色 AWS 帳戶。受管政策包括 AWS 受管政策和客戶受管政策。如需了解如何在受管政策及內嵌政策之間選擇，請參閱《IAM 使用者指南》中的在受管政策和內嵌政策間選擇。

資源型政策

資源型政策是連接到資源的 JSON 政策文件。資源型政策的最常見範例是 IAM 角色信任政策和 Amazon S3 儲存貯體政策。在支援資源型政策的服務中，服務管理員可以使用它們來控制對特定資源的存取權限。對於附加政策的資源，政策會定義指定的主體可以對該資源執行的動作以及在何種條件下執行的動作。您必須在資源型政策中指定主體。委託人可以包含帳戶、使用者、角色、聯合身分使用者或 AWS 服務。

資源型政策是位於該服務中的內嵌政策。您無法在資源型政策中使用來自 IAM 的 AWS 受管政策。

存取控制清單 (ACL)

存取控制清單 (ACL) 可控制哪些主體 (帳戶成員、使用者或角色) 擁有存取某資源的許可。ACL 類似於資源型政策，但它們不使用 JSON 政策文件格式。

Amazon S3 AWS WAF和 Amazon VPC 是支援 ACLs的服務範例。如需進一步了解 ACL，請參閱 Amazon Simple Storage Service 開發人員指南中的存取控制清單 (ACL) 概觀。

其他政策類型

AWS 支援其他較不常見的政策類型。這些政策類型可設定較常見政策類型授予您的最大許可。

- 許可界限 – 許可範圍是一種進階功能，可供您設定身分型政策能授予 IAM 實體 (IAM 使用者或角色) 的最大許可。您可以為實體設定許可界限。所產生的許可會是實體的身分型政策和其許可界限的交集。會在 `Principal` 欄位中指定使用者或角色的資源型政策則不會受到許可界限限制。所有這類政策中的明確拒絕都會覆寫該允許。如需許可界限的詳細資訊，請參閱 IAM 使用者指南中的 IAM 實體許可界限。
- 服務控制政策 SCPs) – SCPs是 JSON 政策，可指定 in. 中組織或組織單位 (OU) 的最大許可 AWS Organizations。 AWS Organizations 是一種用於分組和集中管理您企業擁有 AWS 帳戶 之多個 的服務。若您啟用組織中的所有功能，您可以將服務控制政策 (SCP) 套用到任何或所有帳戶。SCP 會限制成員帳戶中實體的許可，包括每個實體 AWS 帳戶根使用者。如需 Organizations 和 SCP 的詳細資訊，請參閱《AWS Organizations 使用者指南》中的服務控制政策。
- 資源控制政策 (RCP) - RCP 是 JSON 政策，可用來設定您帳戶中資源的可用許可上限，採取這種方式就不需要更新附加至您所擁有的每個資源的 IAM 政策。RCP 會限制成員帳戶中資源的許可，並可能影響身分的有效許可，包括 AWS 帳戶根使用者，無論它們是否屬於您的組織。如需 Organizations 和 RCPs的詳細資訊，包括支援 RCPs AWS 服務 的 清單，請參閱AWS Organizations 《 使用者指南》中的資源控制政策 (RCPs)。
- 工作階段政策 – 工作階段政策是一種進階政策，您可以在透過撰寫程式的方式建立角色或聯合使用者的暫時工作階段時，做為參數傳遞。所產生工作階段的許可會是使用者或角色的身分型政策和工作階段政策的交集。許可也可以來自資源型政策。所有這類政策中的明確拒絕都會覆寫該允許。如需詳細資訊，請參閱 IAM 使用者指南中的工作階段政策。

多種政策類型

將多種政策類型套用到請求時，其結果形成的許可會更為複雜、更加難以理解。若要了解如何 AWS 在涉及多種政策類型時決定是否允許請求，請參閱《IAM 使用者指南》中的政策評估邏輯。

AWS X-Ray 如何使用 IAM

在使用 IAM 管理 X-Ray 的存取權之前，您應該先了解哪些 IAM 功能可與 X-Ray 搭配使用。若要全面了解 X-Ray 和其他 AWS 服務 如何使用 IAM，請參閱《AWS 服務 IAM 使用者指南》中的該使用 IAM。

您可以使用 AWS Identity and Access Management (IAM) 將 X-Ray 許可授予您帳戶中的使用者和運算資源。IAM 控制對 API 層級 X-Ray 服務的存取，以統一地強制執行許可，無論您的使用者使用哪個用戶端 （ 主控台、 AWS SDK AWS CLI)。

若要使用 X-Ray 主控台來檢視追蹤地圖和區段，您只需要讀取許可。若要啟用主控台存取，請將 `AWSXrayReadOnlyAccess` 受管政策新增到您的 IAM 使用者。

針對本機開發和測試，建立具有讀取和寫入許可的 IAM 角色。擔任角色並存放角色的臨時登入資料。您可以搭配 X-Ray 協助程式、 AWS CLI和 AWS SDK 使用這些登入資料。如需詳細資訊，請參閱搭配使用臨時安全登入 AWS CLI資料。

若要將受檢測的應用程式部署到 AWS ，請建立具有寫入許可的 IAM 角色，並將其指派給執行您應用程式的資源。 `AWSXRayDaemonWriteAccess`包含上傳追蹤的許可，以及一些讀取許可，以支援使用抽樣規則。

讀取和寫入政策不包含設定加密金鑰設定及抽樣規則的許可。使用 `AWSXrayFullAccess` 存取這些設定，或是在自訂政策中新增組態 API。針對使用您所建立客戶受管金鑰的加密和解密，您也需要使用金鑰的許可。

主題

- X-Ray 身分型政策
- X-Ray 資源型政策
- 以 X-Ray 標籤為基礎的授權
- 於本機執行您的應用程式
- 在 中執行您的應用程式 AWS
- 用於加密的使用者許可

X-Ray 身分型政策

使用 IAM 身分型政策，您可以指定允許或拒絕的動作和資源，以及在何種條件下允許或拒絕動作。X-Ray 支援特定動作、資源和條件索引鍵。若要了解您在 JSON 政策中使用的所有元素，請參閱 IAM 使用者指南中的 JSON 政策元素參考。

動作

管理員可以使用 AWS JSON 政策來指定誰可以存取內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

JSON 政策的 `Action` 元素描述您可以用來允許或拒絕政策中存取的動作。政策動作通常具有與相關聯 AWS API 操作相同的名稱。有一些例外狀況，例如沒有相符的 API 操作的僅限許可動作。也有一些作業需要政策中的多個動作。這些額外的動作稱為相依動作。

政策會使用動作來授予執行相關聯動作的許可。

X-Ray 中的政策動作在動作之前使用下列字首：`xray:`。例如，若要授予某人使用 X-Ray `GetGroup` API 操作擷取群組資源詳細資訊的許可，請在其政策中包含 `xray:GetGroup`動作。政策陳述式必須包含 `Action` 或 `NotAction` 元素。X-Ray 會定義自己的一組動作，描述您可以使用此服務執行的任務。

若要在單一陳述式中指定多個動作，請用逗號分隔，如下所示：

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

您也可以使用萬用字元 (*) 來指定多個動作。例如，若要指定開頭是 `Get` 文字的所有動作，請包含以下動作：

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

若要查看 X-Ray 動作清單，請參閱《IAM 使用者指南》中的 定義的動作 AWS X-Ray。

資源

管理員可以使用 AWS JSON 政策來指定誰可以存取內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

`Resource` JSON 政策元素可指定要套用動作的物件。陳述式必須包含 `Resource` 或 `NotResource` 元素。最佳實務是使用其 Amazon Resource Name (ARN) 來指定資源。您可以針對支援特定資源類型的動作 (稱為資源層級許可) 來這麼做。

對於不支援資源層級許可的動作 (例如列出操作)，請使用萬用字元 (*) 來表示陳述式適用於所有資源。

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

您可以使用 IAM 政策來控制對 資源的存取。針對支援資源層級許可的動作，您可以使用 Amazon Resource Name (ARN) 來識別要套用政策的資源。

所有 X-Ray 動作都可以在 IAM 政策中使用，以授予或拒絕使用者使用該動作的許可。不過，並非所有 X-Ray 動作都支援資源層級許可，這可讓您指定可執行動作的資源。

對於不支援資源層級許可的動作，您必須使用 "*" 做為資源。

下列 X-Ray 動作支援資源層級許可：

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

以下是 CreateGroup 動作的身分型許可政策範例。此範例示範使用與有唯一 ID 之群組名稱 local-users 有關的 ARN 做為萬用字元。群組建立時產生的唯一 ID，所以無法在政策中事先預測。使用 GetGroup、UpdateGroup 或 DeleteGroup 時，您可以將此定義為萬用字元或確切的 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/*"
            ]
        }
    ]
}
```

以下是 CreateSamplingRule 動作的身分型許可政策範例。

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

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

> **Note**
>
> 依其名稱定義之取樣規則的 ARN。不像群組 ARN，取樣規則沒有唯一產生的 ID。

若要查看 X-Ray 資源類型及其 ARNs 的清單，請參閱《IAM 使用者指南》中的 定義的資源 AWS X-Ray。若要了解您可以使用哪些動作指定每個資源的 ARN，請參閱 AWS X-Ray定義的動作。

條件索引鍵

X-Ray 不提供任何服務特定的條件金鑰，但支援使用一些全域條件金鑰。若要查看所有 AWS 全域條件金鑰，請參閱《IAM 使用者指南》中的AWS 全域條件內容金鑰。

範例

若要檢視 X-Ray 身分型政策的範例，請參閱 AWS X-Ray 身分型政策範例。

X-Ray 資源型政策

X-Ray 支援目前和未來 AWS 服務 整合的資源型政策，例如 Amazon SNS 主動追蹤。X-Ray 資源型政策可由其他 AWS Management Console更新，或透過 AWS SDK 或 CLI 更新。例如，Amazon SNS 主控台會嘗試自動設定資源型政策，以將追蹤傳送至 X-Ray。下列政策文件提供手動設定 X-Ray 資源型政策的範例。

Example Amazon SNS 主動追蹤的 X-Ray 資源型政策範例

此範例政策文件指定 Amazon SNS 將追蹤資料傳送至 X-Ray 所需的許可：

```
{
    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 建立資源型政策，提供 Amazon SNS 將追蹤資料傳送至 X-Ray 的許可：

```
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" } } } ] }'
```

若要使用這些範例，請將 *partition*、*account-id*、 *region*和 取代*topic-name*為您的特定 AWS 分割區、區域、帳戶 ID 和 Amazon SNS 主題名稱。若要授予所有 Amazon SNS 主題將追蹤資料傳送至 X-Ray 的許可，請將主題名稱取代為 *。

以 X-Ray 標籤為基礎的授權

您可以將標籤連接至 X-Ray 群組或取樣規則，或在請求中將標籤傳遞至 X-Ray。如需根據標籤控制存取，請使用 xray:ResourceTag/*key-name*、aws:RequestTag/*key-name* 或 aws:TagKeys 條件索引鍵，在政策的條件元素中，提供標籤資訊。如需標記 X-Ray 資源的詳細資訊，請參閱 標記 X-Ray 取樣規則和群組。

若要檢視身分型政策範例，以根據該資源上的標籤來限制存取資源，請參閱根據標籤管理對 X-Ray 群組和抽樣規則的存取。

於本機執行您的應用程式

經檢測的應用程式會將追蹤資料傳送至 X-Ray 協助程式。協助程式會緩衝區段文件，並分批上傳至 X-Ray 服務。協助程式需要寫入許可，才能將追蹤資料和遙測上傳到 X-Ray 服務。

當您在本機執行 協助程式時，請建立 IAM 角色、擔任該角色，並將臨時登入資料存放在環境變數中，或存放在名為 的檔案中，`credentials`該檔案位於使用者資料夾中名為 `.aws`的資料夾中。如需詳細資訊，請參閱搭配 使用臨時安全登入 AWS CLI資料。

Example ~/.aws/credentials

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

如果您已設定登入資料以搭配 AWS SDK 或 使用 AWS CLI，協助程式可以使用這些登入資料。若有多個可用的描述檔，精靈會使用預設描述檔。

在 中執行您的應用程式 AWS

當您在 上執行應用程式時 AWS，請使用角色將許可授予執行協助程式的 Amazon EC2 執行個體或 Lambda 函數。

- Amazon Elastic Compute Cloud (Amazon EC2) – 建立 IAM 角色，並將其做為執行個體描述檔連接至 EC2 執行個體。
- Amazon Elastic Container Service (Amazon ECS) – 建立 IAM 角色，並將其做為容器執行個體 IAM 角色連接至容器執行個體。
- AWS Elastic Beanstalk (Elastic Beanstalk) – Elastic Beanstalk 在其預設執行個體描述檔中包含 X-Ray 許可。您可以使用預設執行個體描述檔，或是將寫入許可新增到自訂執行個體描述檔。
- AWS Lambda (Lambda) – 將寫入許可新增至函數的執行角色。

建立要與 X-Ray 搭配使用的角色

1. 開啟 IAM 主控台。
2. 選擇角色。

3. 選擇 Create New Role (建立新角色)。
4. 在 Role Name (角色名稱) 中，輸入 **`xray-application`**。選擇 Next Step (後續步驟)。
5. 針對 Role Type (角色類型)，選擇 Amazon EC2。
6. 連接下列 受管政策，讓您的應用程式存取 AWS 服務：

   - AWSXRayDaemonWriteAccess – 授予 X-Ray 協助程式上傳追蹤資料的許可。

   如果您的應用程式使用 AWS SDK 存取其他 服務，請新增授予這些服務存取權的政策。
7. 選擇 Next Step (後續步驟)。
8. 選擇建立角色。

用於加密的使用者許可

根據預設，X-Ray 會加密所有追蹤資料和 ，而且您可以將其設定為使用您管理的金鑰。如果您選擇 AWS Key Management Service 客戶受管金鑰，則需要確保金鑰的存取政策允許您授予 X-Ray 使用它進行加密的許可。您帳戶中的其他使用者也需要存取 金鑰，才能在 X-Ray 主控台中檢視加密的追蹤資料。

對於客戶受管金鑰，請使用允許下列動作的存取政策來設定金鑰：

- 在 X-Ray 中設定金鑰的使用者具有呼叫 `kms:CreateGrant`和 的許可`kms:DescribeKey`。
- 能夠存取加密追蹤資料的使用者必須具備呼叫 `kms:Decrypt` 的許可。

當您在 IAM 主控台的金鑰組態區段中將使用者新增至金鑰使用者群組時，他們具有這兩個操作的許可。只需要在金鑰政策上設定許可，因此您不需要對使用者、群組或角色的任何 AWS KMS 許可。如需詳細資訊，請參閱《 AWS KMS 開發人員指南》中的使用金鑰政策。

對於預設加密，或者如果您選擇 AWS 受管 CMK (`aws/xray`)，許可取決於誰可以存取 X-Ray APIs。任何擁有 `PutEncryptionConfig` 存取權的人員 (包含在 `AWSXrayFullAccess` 中) 都可以變更加密組態。若要防止使用者變更加密金鑰，請不要給予他們使用 `PutEncryptionConfig` 的許可。

AWS X-Ray 身分型政策範例

根據預設，使用者和角色沒有建立或修改 X-Ray 資源的許可。他們也無法使用 AWS Management Console AWS CLI或 AWS API 執行任務。管理員必須建立 IAM 政策，授與使用者和角色在指定資源上執行特定 API 操作所需的許可。管理員接著必須將這些政策連接至需要這些許可的使用者或群組。

若要了解如何使用這些範例 JSON 政策文件建立 IAM 身分型政策，請參閱 IAM 使用者指南中的在 JSON 索引標籤上建立政策。

主題

政策最佳實務

身分型政策會判斷您帳戶中的某個人員是否可以建立、存取或刪除 X-Ray 資源。這些動作可能會讓您的 AWS 帳戶產生費用。當您建立或編輯身分型政策時，請遵循下列準則及建議事項：

- 開始使用 AWS 受管政策並邁向最低權限許可 – 若要開始將許可授予您的使用者和工作負載，請使用將許可授予許多常見使用案例的 AWS 受管政策。它們可在您的 中使用 AWS 帳戶。我們建議您定義特定於使用案例 AWS 的客戶受管政策，以進一步減少許可。如需更多資訊，請參閱 IAM 使用者指南中的 AWS 受管政策或任務職能的AWS 受管政策。
- 套用最低權限許可 – 設定 IAM 政策的許可時，請僅授予執行任務所需的許可。為實現此目的，您可以定義在特定條件下可以對特定資源採取的動作，這也稱為最低權限許可。如需使用 IAM 套用許可的更多相關資訊，請參閱 IAM 使用者指南中的 IAM 中的政策和許可。
- 使用 IAM 政策中的條件進一步限制存取權 – 您可以將條件新增至政策，以限制動作和資源的存取。例如，您可以撰寫政策條件，指定必須使用 SSL 傳送所有請求。如果透過特定 等使用服務動作 AWS 服務，您也可以使用條件來授予其存取權 AWS CloudFormation。如需詳細資訊，請參閱 IAM 使用者指南中的 IAM JSON 政策元素：條件。
- 使用 IAM Access Analyzer 驗證 IAM 政策，確保許可安全且可正常運作 – IAM Access Analyzer 驗證新政策和現有政策，確保這些政策遵從 IAM 政策語言 (JSON) 和 IAM 最佳實務。IAM Access Analyzer 提供 100 多項政策檢查及切實可行的建議，可協助您撰寫安全且實用的政策。如需詳細資訊，請參閱《IAM 使用者指南》中的使用 IAM Access Analyzer 驗證政策。
- 需要多重要素驗證 (MFA) – 如果您的案例需要 IAM 使用者或 中的根使用者 AWS 帳戶，請開啟 MFA 以提高安全性。如需在呼叫 API 操作時請求 MFA，請將 MFA 條件新增至您的政策。如

需詳細資訊，請參閱《IAM 使用者指南》https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_mfa_configure-api-require.html中的透過 MFA 的安全 API 存取。

如需 IAM 中最佳實務的相關資訊，請參閱 IAM 使用者指南中的 IAM 安全最佳實務。

使用 X-Ray 主控台

若要存取 AWS X-Ray 主控台，您必須擁有一組最低的許可。這些許可必須允許您列出和檢視 中 X-Ray 資源的詳細資訊 AWS 帳戶。如果您建立比最基本必要許可更嚴格的身分型政策，則對於具有該政策的實體 (使用者或角色) 而言，主控台就無法如預期運作。

為了確保這些實體仍可使用 X-Ray 主控台，請將 AWSXRayReadOnlyAccess AWS 受管政策連接至實體。此政策在 X-Ray 的 IAM 受管政策中詳細說明。如需詳細資訊，請參閱《IAM 使用者指南》中的新增許可到使用者。

對於僅呼叫 AWS CLI 或 AWS API 的使用者，您不需要允許最低主控台許可。反之，只需允許存取符合您嘗試執行之 API 操作的動作就可以了。

允許使用者檢視他們自己的許可

此範例會示範如何建立政策，允許 IAM 使用者檢視附加到他們使用者身分的內嵌及受管政策。此政策包含在主控台上完成此動作的許可，或使用 AWS CLI 或 AWS API 以程式設計方式完成此動作的許可。

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

根據標籤管理對 X-Ray 群組和抽樣規則的存取

您可以在身分型政策中使用條件，根據標籤控制對 X-Ray 群組和抽樣規則的存取。下列範例政策可用來拒絕使用者角色使用標籤或 建立、刪除`stage:prod`或更新群組的許可`stage:preprod`。如需標記 X-Ray 取樣規則和群組的詳細資訊，請參閱 標記 X-Ray 取樣規則和群組。

若要拒絕使用者建立、更新或刪除具有標籤或 `stage:prod` 的群組`stage:preprod`，請為使用者指派具有類似下列政策的角色。

```
{
    "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"
                    ]
                }
            }
        }
    ]
}
```

若要拒絕建立抽樣規則，請使用 aws:RequestTag 來指示無法作為建立請求的一部分傳遞的標籤。若要拒絕更新或刪除抽樣規則，請使用 aws:ResourceTag來拒絕根據這些資源上的標籤採取的動作。

```
{
    "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"
                    ]
                }
            }
        }
    ]
}
```

您可以將這些政策（或將其合併為單一政策，然後連接政策）連接到您帳戶中的使用者。若要讓使用者變更群組或取樣規則，群組或取樣規則不得加上標籤 `stage=prepod`或 `stage=prod`。條件標籤鍵 `Stage` 符合 `Stage` 和 `stage`，因為條件索引鍵名稱不區分大小寫。如需 條件區塊的詳細資訊，請參閱《IAM 使用者指南》中的 IAM JSON 政策元素：條件。

具有下列政策連接之角色的使用者無法將標籤新增至`role:admin`資源，也無法從與其`role:admin`相關聯的資源移除標籤。

```
{
    "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 受管政策

為了簡化授予許可，IAM 支援每個服務的受管政策。服務可以在發行新的 APIs 時，以新的許可更新這些受管政策。 AWS X-Ray 提供用於唯讀、唯寫和管理員使用案例的受管政策。

- `AWSXrayReadOnlyAccess` – 使用 X-Ray 主控台 AWS CLI或 AWS SDK 從 X-Ray API 取得追蹤資料、追蹤地圖、洞見和 X-Ray 組態的讀取許可。包括可觀測性存取管理員 (OAM) `oam:ListSinks`和`oam:ListAttachedSinks`許可，以允許主控台檢視來源帳戶共用的追蹤，作為 CloudWatch 跨帳戶可觀測性的一部分。`BatchGetTraceSummaryById` 和 `GetDistinctTraceGraphs` API 動作並非由您的程式碼呼叫，也不包含在 AWS CLI 和 AWS SDKs中。

```
{
```

```
    "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` – 使用 X-Ray 協助程式 AWS CLI或 AWS SDK 將區段文件和遙測上傳到 X-Ray API 的寫入許可。包含用於取得抽樣規則及報告抽樣結果的讀取許可。

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "xray:PutTraceSegments",
                "xray:PutTelemetryRecords",
                "xray:GetSamplingRules",
                "xray:GetSamplingTargets",
                "xray:GetSamplingStatisticSummaries"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

- AWSXrayCrossAccountSharingConfiguration – 准許建立、管理和檢視可觀測性存取管理員連結，以在帳戶之間共用 X-Ray 資源。用於啟用來源和監控帳戶之間的 CloudWatch 跨帳戶可觀測性。

```
{
    "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 APIs許可，包括讀取許可、寫入許可，以及設定加密金鑰設定和取樣規則的許可。包括可觀測性存取管理員 (OAM) `oam:ListSinks`和`oam:ListAttachedSinks`許可，以允許主控台檢視來源帳戶共用的追蹤，作為 CloudWatch 跨帳戶可觀測性的一部分。

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

新增受管政策連接到 IAM 使用者、群組或角色

1. 開啟 IAM 主控台。
2. 開啟與您的執行個體描述檔、IAM 使用者或 IAM 群組關聯的角色。
3. 在 Permissions (許可) 下，關聯受管政策。

AWS 受管政策的 X-Ray 更新

檢視自此服務開始追蹤這些變更以來，X-Ray AWS 受管政策更新的詳細資訊。如需此頁面變更的自動提醒，請訂閱 X-Ray 文件歷史記錄頁面上的 RSS 摘要。

變更	描述	日期
適用於 X-Ray 的 IAM 受管政策 – 新增了 `AWSXrayCrossAccountSharingConfiguration` 、和更新 `AWSXrayReadOnlyAccess` 和 `AWSXrayFullAccess` 政策。	X-Ray 已將可觀測性存取管理員 (OAM) 許可`oam:ListSinks` 和 `oam:ListAttachedSinks` 新增至這些政策，以允許主控台檢視來源帳戶共用的追蹤，做為 CloudWatch 跨帳戶可觀測性的一部分。	2022 年 11 月 27 日
X-Ray 的 IAM 受管政策 – `AWSXrayReadOnlyAccess` 政策更新。	X-Ray 新增了 API 動作 `ListResourcePolicies` 。	2022 年 11 月 15 日
使用 X-Ray 主控台 – `AWSXrayReadOnlyAccess` 更新政策	X-Ray 新增了兩個新的 API 動作 `BatchGetTraceSummaryById` 和 `GetDistinctTraceGraphs` 。 這些動作並非由您的程式碼呼叫。因此，這些 API 動作不包含在 AWS CLI 和 AWS SDKs 中。	2022 年 11 月 11 日

在 IAM 政策中指定資源

您可以使用 IAM 政策來控制對 資源的存取。針對支援資源層級許可的動作，您可以使用 Amazon Resource Name (ARN) 來識別要套用政策的資源。

所有 X-Ray 動作都可以在 IAM 政策中使用，以授予或拒絕使用者使用該動作的許可。不過，並非所有 X-Ray 動作都支援資源層級許可，這可讓您指定可執行動作的資源。

對於不支援資源層級許可的動作，您必須使用 "*" 做為資源。

下列 X-Ray 動作支援資源層級許可：

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

以下是 `CreateGroup` 動作的身分型許可政策範例。此範例示範使用與有唯一 ID 之群組名稱 `local-users` 有關的 ARN 做為萬用字元。群組建立時產生的唯一 ID，所以無法在政策中事先預測。使用 `GetGroup`、`UpdateGroup` 或 `DeleteGroup` 時，您可以將此定義為萬用字元或確切的 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/*"
            ]
        }
    ]
```

```
}
```

以下是 CreateSamplingRule 動作的身分型許可政策範例。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "xray:CreateSamplingRule"
            ],
            "Resource": [
                "arn:aws:xray:eu-west-1:123456789012:sampling-rule/base-scorekeep"
            ]
        }
    ]
}
```

> **Note**
>
> 依其名稱定義之取樣規則的 ARN。不像群組 ARN，取樣規則沒有唯一產生的 ID。

對 AWS X-Ray 身分和存取進行故障診斷

使用以下資訊來協助您診斷和修正使用 X-Ray 和 IAM 時可能遇到的常見問題。

主題

- 我未獲授權在 X-Ray 中執行動作
- 我未獲授權，不得執行 iam:PassRole
- 我是管理員，想要允許其他人存取 X-Ray
- 我想要允許 以外的人員 AWS 帳戶 存取我的 X-Ray 資源

我未獲授權在 X-Ray 中執行動作

如果 AWS Management Console 告訴您無權執行動作，則必須聯絡管理員尋求協助。您的管理員是為您提供簽署憑證的人員。

當mateojackson使用者嘗試使用 主控台來檢視取樣規則的詳細資訊，但沒有xray:GetSamplingRules許可時，會發生下列範例錯誤。

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

在此情況下，Mateo 請求管理員更新他的政策，以允許他使用 xray:GetSamplingRules 動作來存取取樣規則資源。

我未獲授權，不得執行 iam:PassRole

如果您收到錯誤，告知您無權執行iam:PassRole動作，則必須更新您的政策，以允許您將角色傳遞給 X-Ray。

有些 AWS 服務 可讓您將現有角色傳遞給該服務，而不是建立新的服務角色或服務連結角色。如需執行此作業，您必須擁有將角色傳遞至該服務的許可。

當名為 的 IAM marymajor 使用者嘗試使用主控台在 X-Ray 中執行動作時，會發生下列範例錯誤。但是，動作請求服務具備服務角色授予的許可。Mary 沒有將角色傳遞至該服務的許可。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

在這種情況下，Mary 的政策必須更新，允許她執行 iam:PassRole 動作。

如果您需要協助，請聯絡您的 AWS 管理員。您的管理員提供您的簽署憑證。

我是管理員，想要允許其他人存取 X-Ray

若要允許其他人存取 X-Ray，您必須將許可授予需要存取的人員或應用程式。如果您使用 AWS IAM Identity Center 來管理人員和應用程式，您可以將許可集指派給使用者或群組，以定義其存取層級。許可集會自動建立 IAM 政策，並將其指派給與該人員或應用程式相關聯的 IAM 角色。如需詳細資訊，請參閱AWS IAM Identity Center 《 使用者指南》中的許可集。

如果您不是使用 IAM Identity Center，則必須為需要存取的人員或應用程式建立 IAM 實體 （使用者或角色）。然後，您必須將政策連接到實體，以授予他們在 X-Ray 中的正確許可。授予許可後，請將登入資料提供給使用者或應用程式開發人員。他們將使用這些登入資料來存取 AWS。若要進一步了解如何建立 IAM 使用者、群組、政策和許可，請參閱《IAM 使用者指南》中的 IAM 身分和政策和許可。

我想要允許 以外的人員 AWS 帳戶 存取我的 X-Ray 資源

您可以建立一個角色，讓其他帳戶中的使用者或您組織外部的人員存取您的資源。您可以指定要允許哪些信任物件取得該角色。針對支援基於資源的政策或存取控制清單 (ACL) 的服務，您可以使用那些政策來授予人員存取您的資源的許可。

如需進一步了解，請參閱以下內容：

- 若要了解 X-Ray 是否支援這些功能，請參閱 AWS X-Ray 如何使用 IAM。
- 若要了解如何 AWS 帳戶 在您擁有的 資源之間提供存取權，請參閱《IAM 使用者指南》中的在您擁有 AWS 帳戶 的另一個 中提供存取權給 IAM 使用者。
- 若要了解如何將資源的存取權提供給第三方 AWS 帳戶，請參閱《IAM 使用者指南》中的將存取權提供給第三方 AWS 帳戶 擁有的 。
- 如需了解如何透過聯合身分提供存取權，請參閱 IAM 使用者指南中的將存取權提供給在外部進行身分驗證的使用者 (聯合身分)。
- 如需了解使用角色和資源型政策進行跨帳戶存取之間的差異，請參閱《IAM 使用者指南》中的 IAM 中的跨帳戶資源存取。

在 中記錄和監控 AWS X-Ray

監控是維護您 AWS 解決方案之可靠性、可用性和效能的重要部分。您應該從 AWS 解決方案的所有部分收集監控資料，以便在發生多點故障時更輕鬆地偵錯。 AWS 提供數種工具來監控 X-Ray 資源並回應潛在事件：

AWS CloudTrail 日誌

AWS X-Ray 與 整合， AWS CloudTrail 以記錄使用者、角色或 X-Ray 中的 AWS 服務所做的 API 動作。您可以使用 CloudTrail 即時監控 X-Ray API 請求，並將日誌存放在 Amazon S3、Amazon CloudWatch Logs 和 Amazon CloudWatch Events 中。如需詳細資訊，請參閱使用 記錄 X-Ray API 呼叫 AWS CloudTrail。

AWS Config 追蹤

AWS X-Ray 與 整合 AWS Config ，以記錄對 X-Ray 加密資源所做的組態變更。您可以使用 AWS Config 清查 X-Ray 加密資源、稽核 X-Ray 組態歷史記錄，並根據資源變更傳送通知。如需詳細資訊，請參閱使用 追蹤 X-Ray 加密組態變更 AWS Config。

Amazon CloudWatch 監控

您可以使用適用於 Java 的 X-Ray 開發套件，從收集的 X-Ray 區段發佈未取樣的 Amazon CloudWatch 指標。這些指標衍生自區段的開始和結束時間，以及錯誤、故障和節流狀態標記。您可以使用這些追蹤指標，公開子區段內的重試和相依性問題。如需詳細資訊，請參閱AWS X-Ray 適用於 Java 的 X-Ray 開發套件的 指標。

的合規驗證 AWS X-Ray

若要了解 AWS 服務 是否在特定合規計劃範圍內，請參閱AWS 服務 合規計劃範圍內的 ，並選擇您感興趣的合規計劃。如需一般資訊，請參閱AWS 合規計劃。

您可以使用 下載第三方稽核報告 AWS Artifact。如需詳細資訊，請參閱下載 中的 AWS Artifact報告。

使用 時的合規責任 AWS 服務 取決於資料的機密性、您公司的合規目標，以及適用的法律和法規。AWS 提供下列資源以協助合規：

- 安全合規與治理 - 這些解決方案實作指南內容討論了架構考量，並提供部署安全與合規功能的步驟。
- HIPAA 合格服務參考 – 列出 HIPAA 合格服務。並非所有 AWS 服務 都符合 HIPAA 資格。
- AWS 合規資源 – 此工作手冊和指南集合可能適用於您的產業和位置。
- AWS 客戶合規指南 – 透過合規的角度了解共同責任模型。本指南摘要說明保護 的最佳實務， AWS 服務 並將指引映射至跨多個架構的安全控制 （ 包括國家標準技術研究所 (NIST)、支付卡產業安全標準委員會 (PCI) 和國際標準化組織 (ISO))。
- 《 AWS Config 開發人員指南》中的使用 規則評估資源 – AWS Config 服務會評估資源組態符合內部實務、產業準則和法規的程度。
- AWS Security Hub – 這 AWS 服務 可讓您全面檢視其中的安全狀態 AWS。Security Hub 使用安全控制，可評估您的 AWS 資源並檢查您的法規遵循是否符合安全業界標準和最佳實務。如需支援的服務和控制清單，請參閱「Security Hub 控制參考」。
- Amazon GuardDuty – 這會監控您的環境是否有可疑和惡意活動，以 AWS 服務 偵測對您 AWS 帳戶、工作負載、容器和資料的潛在威脅。GuardDuty 可滿足特定合規架構所規定的入侵偵測需求，以協助您因應 PCI DSS 等各種不同的合規需求。
- AWS Audit Manager – 這 AWS 服務 可協助您持續稽核 AWS 用量，以簡化您管理風險的方式，以及是否符合法規和業界標準。

中的彈性 AWS X-Ray

AWS 全球基礎設施是以 AWS 區域 和 可用區域為基礎建置。 AWS 區域 提供多個實體分隔和隔離的可用區域，這些區域與低延遲、高輸送量和高備援聯網連接。透過可用區域，您所設計與操作的應用程式和資料庫，就能夠在可用區域之間自動容錯移轉，而不會發生中斷。可用區域的可用性、容錯能力和擴充能力，均較單一或多個資料中心的傳統基礎設施還高。

如需 AWS 區域 和可用區域的詳細資訊，請參閱 AWS 全球基礎設施。

中的基礎設施安全 AWS X-Ray

作為受管服務， AWS X-Ray 受到 AWS 全球網路安全的保護。如需 AWS 安全服務以及 如何 AWS 保護基礎設施的相關資訊，請參閱AWS 雲端安全。若要使用基礎設施安全的最佳實務來設計您的 AWS 環境，請參閱安全支柱 AWS Well-Architected Framework 中的基礎設施保護。

您可以使用 AWS 已發佈的 API 呼叫，透過網路存取 X-Ray。使用者端必須支援下列專案：

- Transport Layer Security (TLS)。我們需要 TLS 1.2 並建議使用 TLS 1.3。
- 具備完美轉送私密(PFS)的密碼套件，例如 DHE (Ephemeral Diffie-Hellman)或 ECDHE (Elliptic Curve Ephemeral Diffie-Hellman)。現代系統(如 Java 7 和更新版本)大多會支援這些模式。

此外，請求必須使用存取金鑰 ID 和與 IAM 主體相關聯的私密存取金鑰來簽署。或者，您可以使用 AWS Security Token Service (AWS STS) 以產生暫時安全憑證以簽署請求。

AWS X-Ray 搭配 VPC 端點使用

如果您使用 Amazon Virtual Private Cloud (Amazon VPC) 託管 AWS 資源，您可以在 VPC 和 X-Ray 之間建立私有連線。這可讓 Amazon VPC 中的資源與 X-Ray 服務通訊，而無需透過公有網際網路。

Amazon VPC 是 AWS 服務 ，可用來在您定義的虛擬網路中啟動 AWS 資源。您可利用 VPC 來控制您的網路設定，例如 IP 地址範圍、子網路、路由表和網路閘道。若要將 VPC 連線至 X-Ray，您可以定義介面 VPC 端點。端點為 X-Ray 提供可靠、可擴展的連線，而不需要網際網路閘道、網路位址轉譯 (NAT) 執行個體或 VPN 連接。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的什麼是 Amazon VPC。

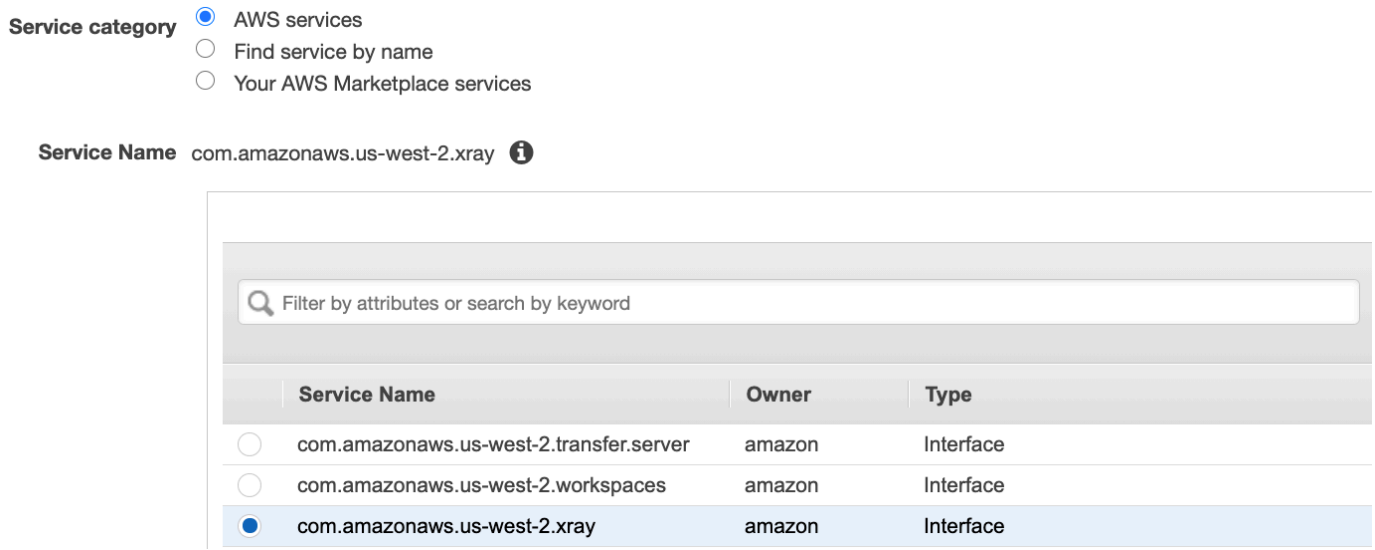
介面 VPC 端點採用 AWS PrivateLink 技術，這項 AWS 技術 AWS 服務 可透過使用具有私有 IP 地址的彈性網路介面，在 之間進行私有通訊。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的 New – AWS PrivateLink for AWS 服務 部落格文章和入門。

為了確保您可以在所選的 X-Ray 中建立 VPC 端點 AWS 區域，請參閱 支援地區。

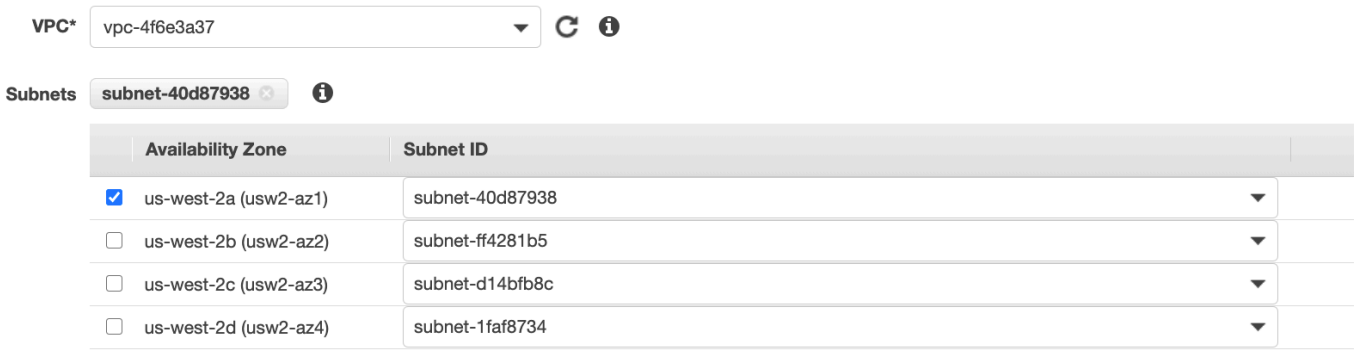
為 X-Ray 建立 VPC 端點

若要開始搭配 VPC 使用 X-Ray，請為 X-Ray 建立介面 VPC 端點。

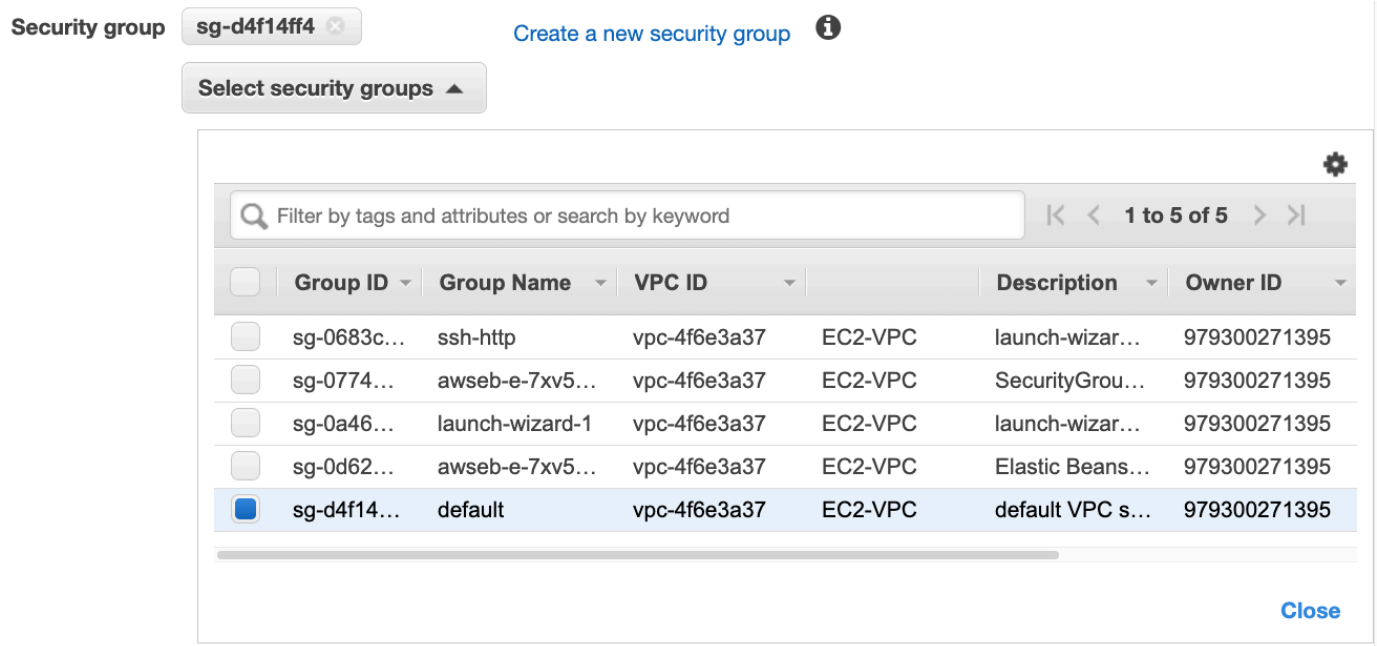
1. 在 https://console.aws.amazon.com/vpc/ 開啟 Amazon VPC 主控台。
2. 在導覽窗格中導覽至端點，然後選擇建立端點。
3. 搜尋並選取 AWS X-Ray 服務的名稱：`com.amazonaws.region.xray`。



4. 選取您想要的 VPC，然後在 VPC 中選取子網路以使用介面端點。端點網路界面會在選取的子網路中建立。您可以指定不同可用區域中的多個子網路 (服務所支援)，協助確保界面端點在可用區域失敗的狀況下保有彈性。如果您這樣做，則會在您指定的每個子網路中建立界面網路界面。



5. （選用）預設會為端點啟用私有 DNS，因此您可以使用其預設 DNS 主機名稱向 X-Ray 提出請求。您可以選擇停用它。
6. 指定要與端點網路界面建立關聯的安全群組。



7. （選用）指定自訂政策以控制存取 X-Ray 服務的許可。根據預設，允許完整存取。

控制對 X-Ray VPC 端點的存取

當您建立或修改端點時，VPC 端點政策是您連接至端點的 IAM 資源政策。如果您未在建立端點時連接政策，Amazon VPC 會以預設政策連接以允許完整存取服務。端點政策不會覆寫或取代 IAM 使用者政策或服務特定的政策。這個另行區分的政策會控制從端點到所指定之服務的存取。端點政策必須以 JSON 格式撰寫。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的使用 VPC 端點控制服務的存取。

VPC 端點政策可讓您控制各種 X-Ray 動作的許可。例如，您可以建立政策，僅允許 PutTraceSegment 並拒絕所有其他動作。這會限制 VPC 中的工作負載和服務僅將追蹤資料傳送至 X-Ray，並拒絕任何其他動作，例如擷取資料、變更加密組態或建立/更新群組。

以下是 X-Ray 端點政策的範例。此政策可讓使用者透過 VPC 連線至 X-Ray，將區段資料傳送至 X-Ray，並防止他們執行其他 X-Ray 動作。

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

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

編輯 X-Ray 的 VPC 端點政策

1. 在 https://console.aws.amazon.com/vpc/ 開啟 Amazon VPC 主控台。
2. 在導覽窗格中選擇 Endpoints (端點)。
3. 如果您尚未建立 X-Ray 的端點，請遵循中的步驟為 X-Ray 建立 VPC 端點。
4. 選取 com.amazonaws.*region*.xray 端點，然後選擇政策索引標籤。
5. 選擇 Edit Policy (編輯政策)，然後進行變更。

支援地區

X-Ray 目前支援下列 VPC 端點 AWS 區域：

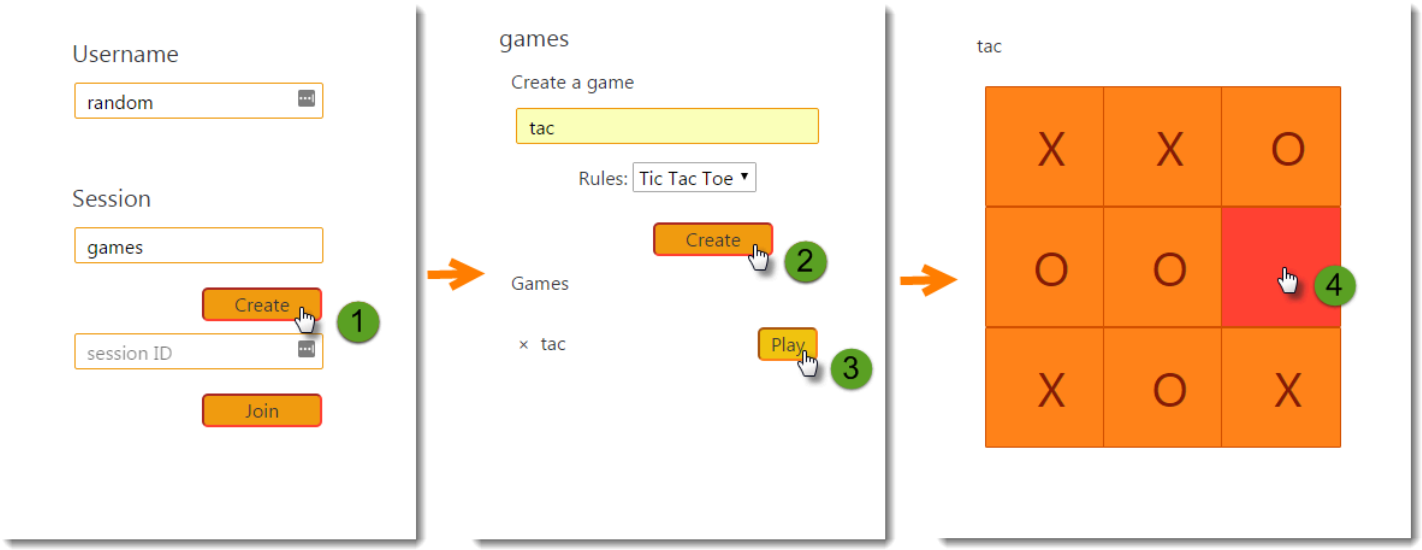
- 美國東部 (俄亥俄)
- 美國東部 (維吉尼亞北部)
- 美國西部 (加利佛尼亞北部)
- 美國西部 (奧勒岡)
- 非洲 (開普敦)
- 亞太區域 (香港)
- 亞太區域 (孟買)
- 亞太區域 (大阪)
- 亞太區域 (首爾)
- 亞太區域 (新加坡)
- 亞太區域 (雪梨)
- 亞太區域 (東京)
- 加拿大 (中部)
- 歐洲 (法蘭克福)
- 歐洲 (愛爾蘭)
- 歐洲 (倫敦)

- 歐洲 (米蘭)
- 歐洲 (巴黎)
- 歐洲 (斯德哥爾摩)
- 中東 (巴林)
- 南美洲 (聖保羅)
- AWS GovCloud （美國東部）
- AWS GovCloud （美國西部）

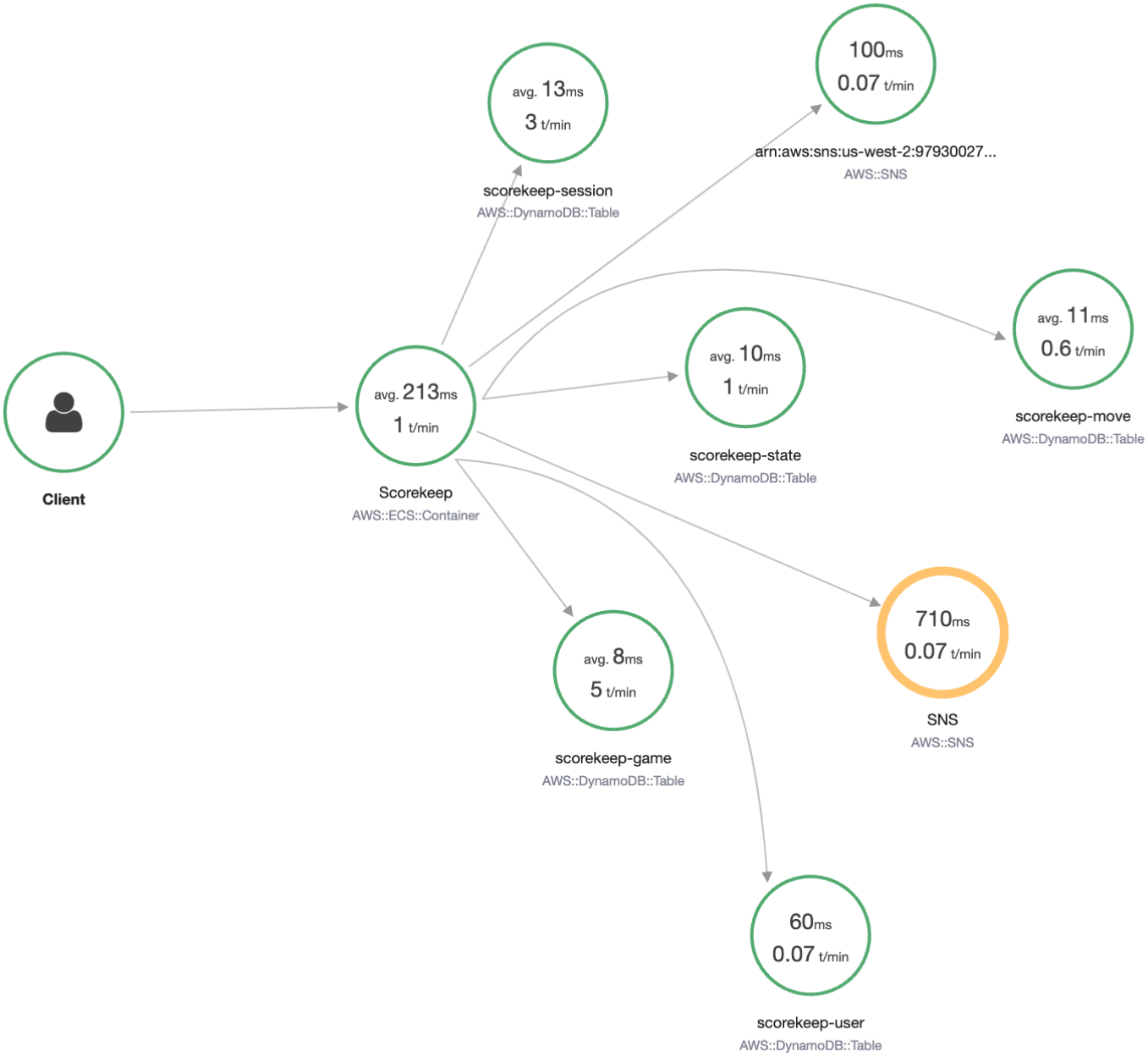
AWS X-Ray 範例應用程式

AWS X-Ray eb-java-scorekeep 範例應用程式可在 GitHub 上取得，顯示使用 AWS X-Ray 開發套件來檢測傳入 HTTP 呼叫、DynamoDB SDK 用戶端和 HTTP 用戶端。範例應用程式使用 AWS CloudFormation 來建立 DynamoDB 資料表、在執行個體上編譯 Java 程式碼，以及執行 X-Ray 協助程式，而不需要任何其他組態。

請參閱 Scorekeep 教學課程，以使用 AWS Management Console 或 開始安裝和使用經檢測的範例應用程式 AWS CLI。



此範例包含前端 Web 應用程式、其呼叫的 API，以及用於存放資料的 DynamoDB 資料表。具有篩選條件、外掛程式和檢測 AWS SDK 用戶端的基本檢測會顯示在專案的`xray-gettingstarted`分支中。這次您在入門教學中部署的項目。由於此分支僅包含基本項目，您可以將它針對 `master` 分支進行 diff，來快速了解基本項目。



範例應用程式會在這些檔案中示範基本檢測：

- HTTP 請求篩選條件 – `WebConfig.java`
- AWS SDK 用戶端檢測 – `build.gradle`

應用程式`xray`分支包括使用 HTTPClient、標註、SQL 查詢、自訂子區段、 檢測的 AWS Lambda函數，以及檢測的初始化程式碼和指令碼。

為了支援使用者登入並在瀏覽器 適用於 JavaScript 的 AWS SDK 中使用 ，`xray-cognito`分支新增了 Amazon Cognito 以支援使用者身分驗證和授權。透過從 Amazon Cognito 擷取的登入資料，Web 應用程式也會將追蹤資料傳送至 X-Ray，以記錄用戶端觀點的請求資訊。瀏覽器用戶端會在追蹤映射上顯示為自己的節點，並記錄其他資訊，包括使用者正在檢視的頁面 URL，以及使用者的 ID。

最後，`xray-worker`分支會新增經過檢測的 Python Lambda 函數，該函數會獨立執行，處理來自 Amazon SQS 佇列的項目。Scorekeep 會在每次遊戲結束時新增項目到佇列中。由 CloudWatch Events 觸發的 Lambda 工作者每隔幾分鐘從佇列中提取項目，並處理它們以將遊戲記錄存放在 Amazon S3 中進行分析。

主題

Scorekeep 範例應用程式入門

本教學課程使用 Scorekeep 範例應用程式的分支，該`xray-gettingstarted`分支使用 AWS CloudFormation 來建立和設定在 Amazon ECS 上執行範例應用程式和 X-Ray 協助程式的資源。應用程式使用 Spring 架構來實作 JSON Web API，並使用 適用於 Java 的 AWS SDK 將資料保留至 Amazon DynamoDB。應用程式中的 servlet 篩選條件會檢測應用程式提供的所有傳入請求，而 AWS SDK 用戶端上的請求處理常式會檢測對 DynamoDB 的下游呼叫。

您可以使用 AWS Management Console 或 遵循本教學課程 AWS CLI。

章節

- 使用 CloudFormation 安裝 Scorekeep 應用程式
- 產生追蹤資料
- 在 中檢視追蹤映射 AWS Management Console
- 設定 Amazon SNS 通知
- 探索範例應用程式
- 選用：最低權限政策
- 清除
- 後續步驟

先決條件

本教學課程使用 AWS CloudFormation 來建立和設定執行範例應用程式和 X-Ray 協助程式的資源。安裝和執行教學課程需要下列先決條件：

1. 如果您使用具有有限許可的 IAM 使用者，請在 IAM 主控台中新增下列使用者政策：
   - `AWSCloudFormationFullAccess` – 存取和使用 CloudFormation
   - `AmazonS3FullAccess` – 使用 將範本檔案上傳至 CloudFormation AWS Management Console
   - `IAMFullAccess` – 建立 Amazon ECS 和 Amazon EC2 執行個體角色
   - `AmazonEC2FullAccess` – 建立 Amazon EC2 資源
   - `AmazonDynamoDBFullAccess` – 建立 DynamoDB 資料表
   - `AmazonECS_FullAccess` – 建立 Amazon ECS 資源
   - `AmazonSNSFullAccess` – 建立 Amazon SNS 主題
   - `AWSXrayReadOnlyAccess` – 用於在 X-Ray 主控台中檢視追蹤映射和追蹤的許可
2. 若要使用 執行教學課程 AWS CLI，請安裝 CLI 2.7.9 版或更新版本，然後與上一個步驟中的使用者設定 CLI。 AWS CLI 使用 使用者設定 時，請確定已設定 區域。如果未設定區域，您將需要附加`--region` *`AWS-REGION`*到每個 CLI 命令。
3. 確保已安裝 Git，以複製範例應用程式儲存庫。
4. 使用以下程式碼範例來複製 Scorekeep 儲存庫的`xray-gettingstarted`分支：

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

使用 CloudFormation 安裝 Scorekeep 應用程式

AWS Management Console

使用 安裝範例應用程式 AWS Management Console

1. 開啟 CloudFormation 主控台
2. 選擇建立堆疊，然後從下拉式功能表中選擇使用新資源。
3. 在 Specify template (指定範本) 區段中，選擇 Upload a template file (上傳範本檔案)。
4. 選取選擇檔案，導覽至複製 git 儲存庫時建立的`xray-scorekeep/cloudformation`資料夾，然後選擇`cf-resources.yaml`檔案。
5. 選擇 Next (下一步) 繼續。
6. 在堆疊名稱文字方塊`scorekeep`中輸入 ，然後選擇頁面底部的下一步以繼續。請注意，本教學課程的其餘部分假設堆疊名為 `scorekeep`。
7. 捲動至設定堆疊選項頁面底部，然後選擇下一步以繼續。
8. 捲動至檢閱頁面底部，選擇確認 CloudFormation 可能會使用自訂名稱建立 IAM 資源的核取方塊，然後選擇建立堆疊。
9. 現在正在建立 CloudFormation 堆疊。堆疊狀態將大約`CREATE_IN_PROGRESS`為五分鐘，然後再變更為 `CREATE_COMPLETE`。狀態會定期重新整理，或者您可以重新整理頁面。

AWS CLI

使用 安裝範例應用程式 AWS CLI

1. 導覽至您先前在教學課程中複製的`xray-scorekeep`儲存庫`cloudformation`資料夾：

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

2. 輸入下列 AWS CLI 命令來建立 CloudFormation 堆疊：

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

3. 等到 CloudFormation 堆疊狀態為 `CREATE_COMPLETE`，這大約需要五分鐘。使用下列 AWS CLI 命令來檢查狀態：

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

產生追蹤資料

範例應用程式包含前端 Web 應用程式。使用 Web 應用程式產生 API 的流量，並將追蹤資料傳送至 X-Ray。首先，使用 AWS Management Console 或 擷取 Web 應用程式 URL AWS CLI：

AWS Management Console

使用 尋找應用程式 URL AWS Management Console

1. 開啟 CloudFormation 主控台
2. 從清單中選擇`scorekeep`堆疊。
3. 選擇`scorekeep`堆疊頁面上的輸出索引標籤，然後選擇 `LoadBalancerUrl` URL 連結以開啟 Web 應用程式。

AWS CLI

使用 尋找應用程式 URL AWS CLI

1. 使用下列命令來顯示 Web 應用程式的 URL：

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

2. 複製此 URL 並在瀏覽器中開啟 以顯示 Scorekeep Web 應用程式。

使用 Web 應用程式產生追蹤資料

1. 選擇 Create (建立) 來產生使用者及工作階段。
2. 輸入 game name (遊戲名稱)、將 Rules (規則) 設為 Tic Tac Toe (井字遊戲)，然後選擇 Create (建立)。
3. 選擇 Play (遊玩) 來啟動遊戲。
4. 選擇一個磚來進行移動並變更遊戲狀態。

每個步驟都會產生對 API 的 HTTP 請求，以及對 DynamoDB 的下游呼叫，以讀取和寫入使用者、工作階段、遊戲、移動和狀態資料。

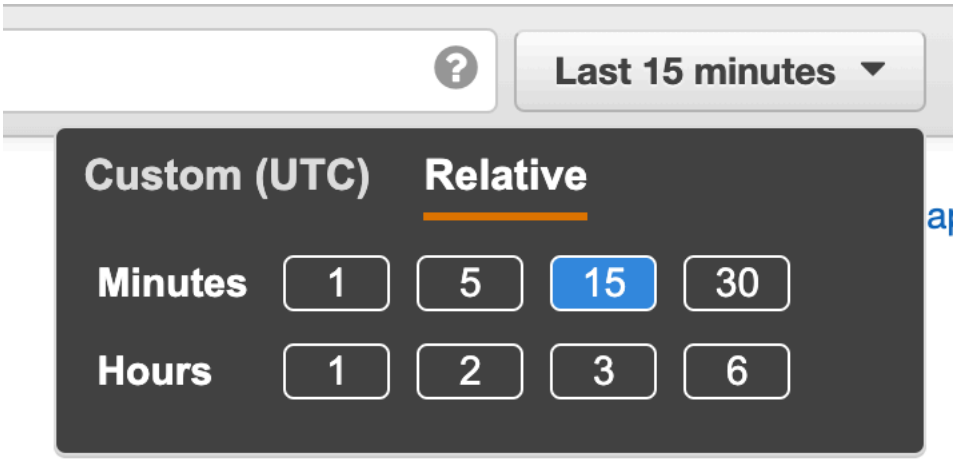
在 中檢視追蹤映射 AWS Management Console

您可以在 X-Ray 和 CloudWatch 主控台中查看範例應用程式產生的追蹤映射和追蹤。

X-Ray console

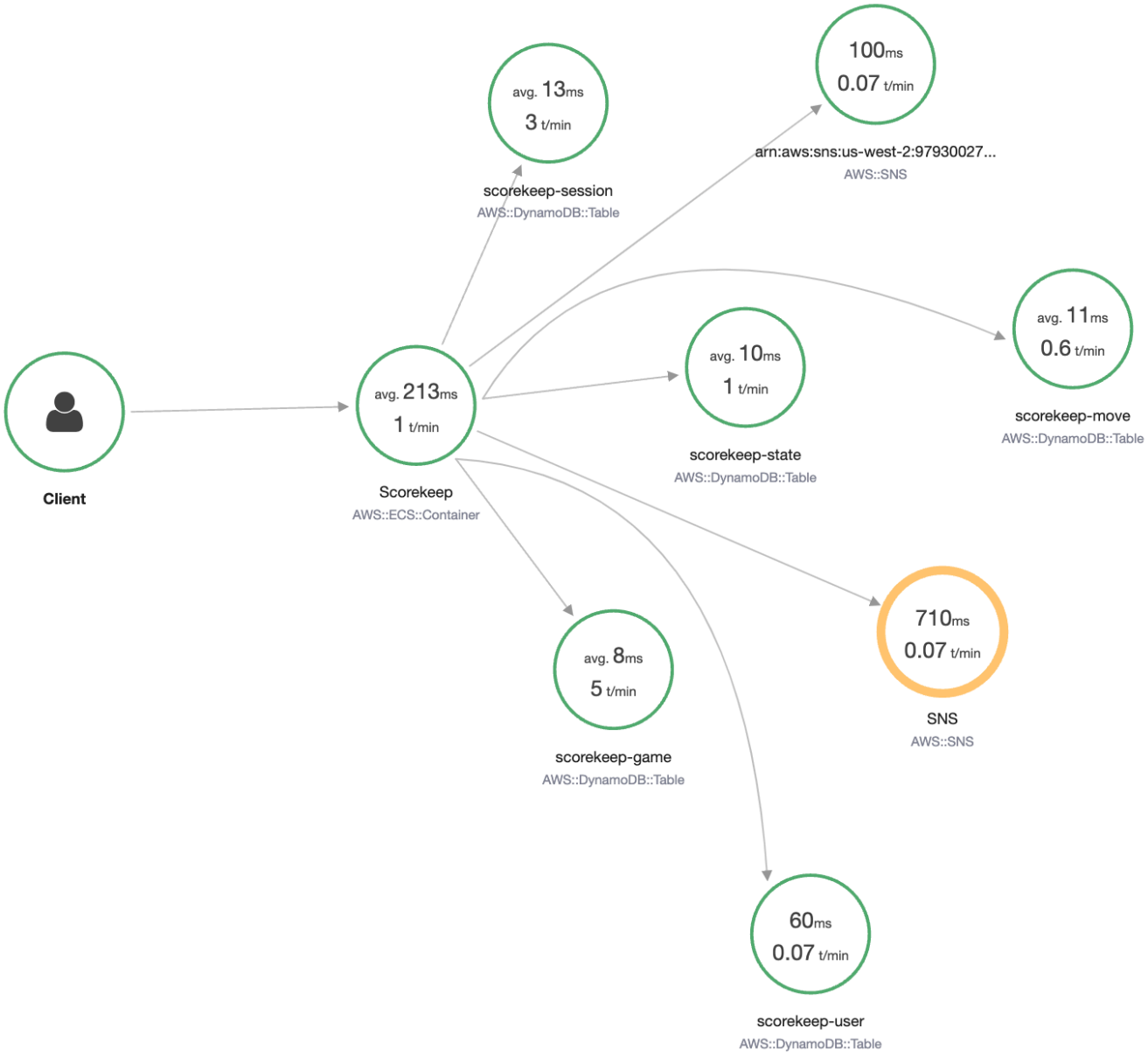
使用 X-Ray 主控台

1. 開啟 X-Ray 主控台的追蹤映射頁面。
2. 主控台會顯示 X-Ray 從應用程式傳送的追蹤資料產生的服務圖表。請務必視需要調整追蹤映射的期間，以確保它在您第一次啟動 Web 應用程式後會顯示所有追蹤。



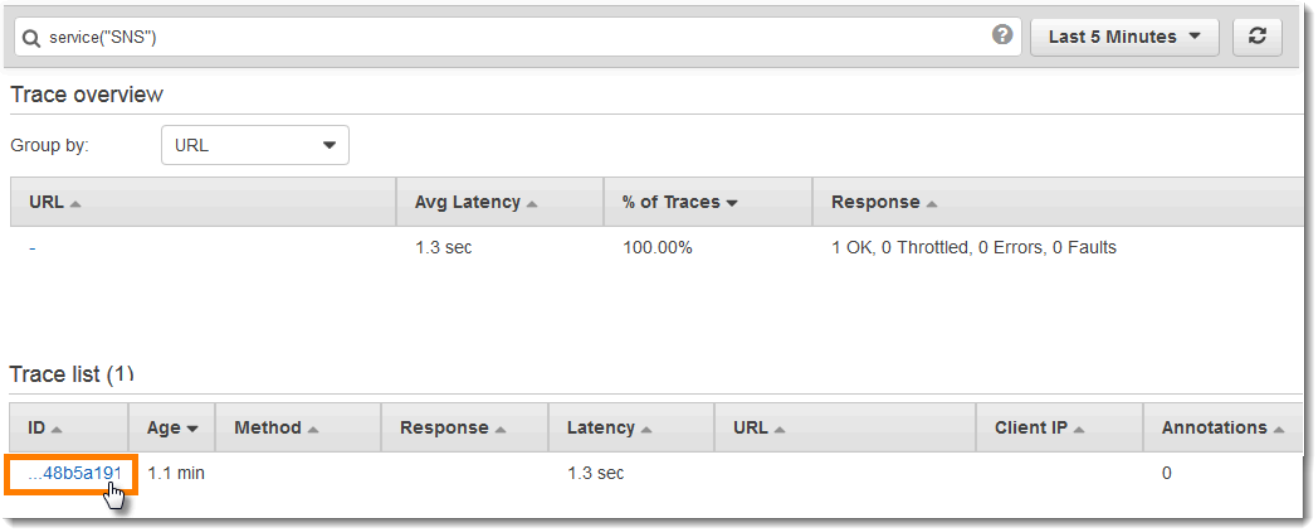
追蹤映射會顯示 Web 應用程式用戶端、在 Amazon ECS 中執行的 API，以及應用程式使用的每個 DynamoDB 資料表。每個向應用程式發出的請求都會在命中 API 時受到追蹤、產生向下游服務發出的請求並完成，其數量上限為可設定的每秒請求數量上限。

您可以在服務圖表中選擇任何節點來檢視產生流量至該節點的請求追蹤。目前，Amazon SNS 節點為黃色。向下切入來了解原因。

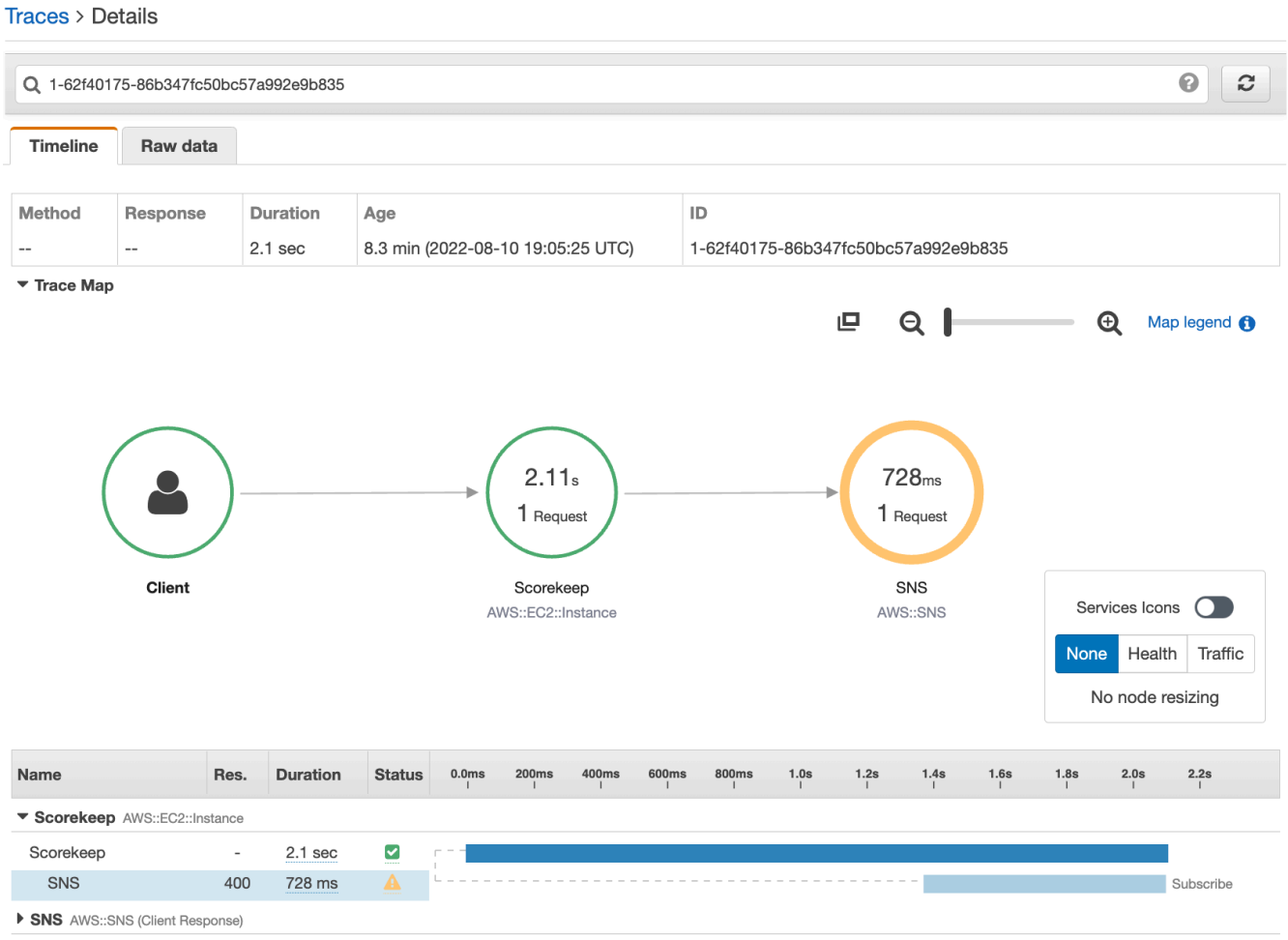


尋找錯誤原因

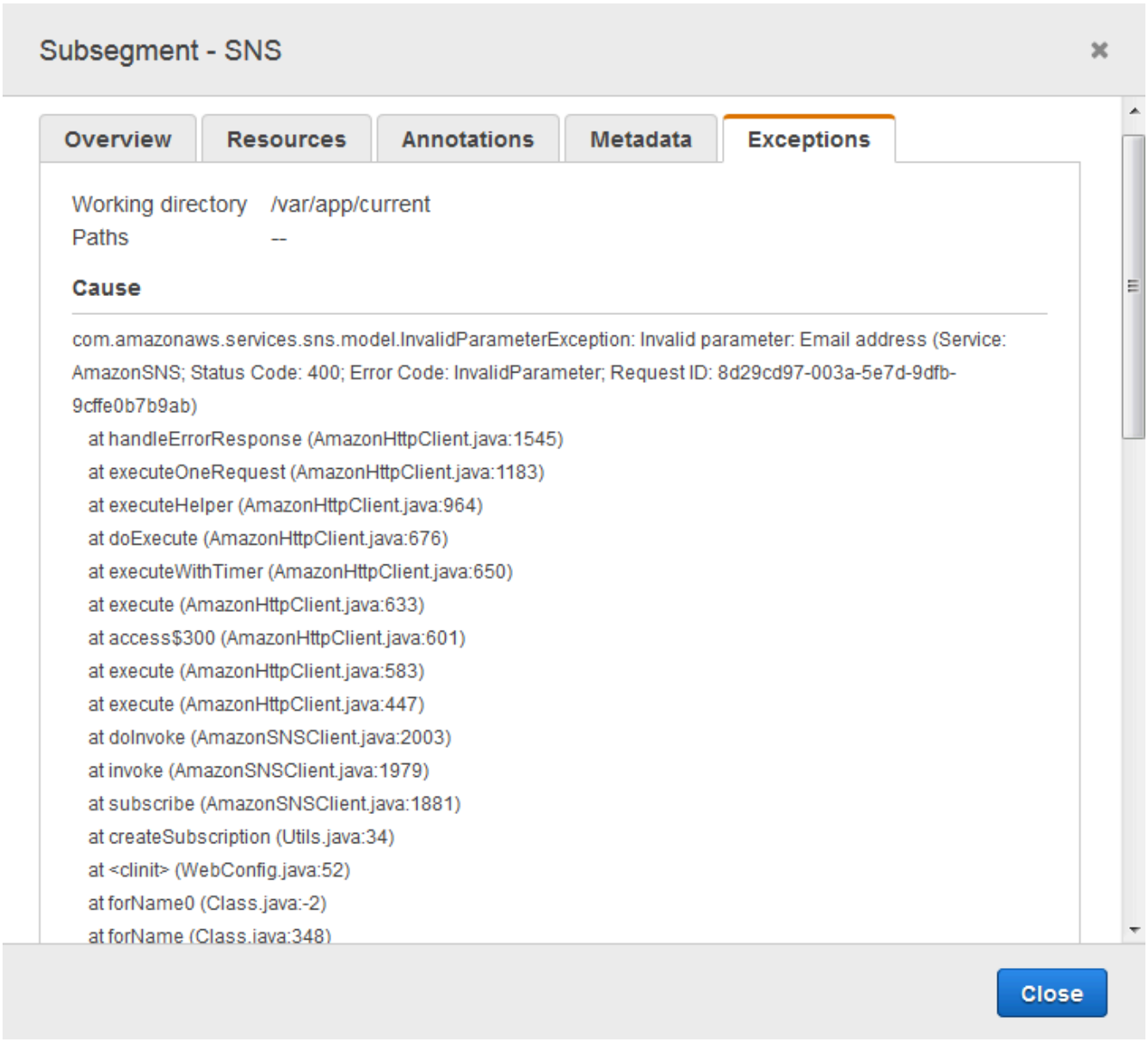
1. 選擇名為 SNS 的節點。節點詳細資訊面板隨即顯示。
2. 選擇 View traces (檢視追蹤) 以存取 Trace overview (追蹤概觀) 畫面。
3. 從 Trace list (追蹤清單) 中選擇追蹤。此追蹤不具有方法或 URL，因為它是在啟動期間，而非回應傳入請求時記錄。



4. 選擇頁面底部的 Amazon SNS 區段中的錯誤狀態圖示，以開啟 SNS 子區段的例外狀況頁面。



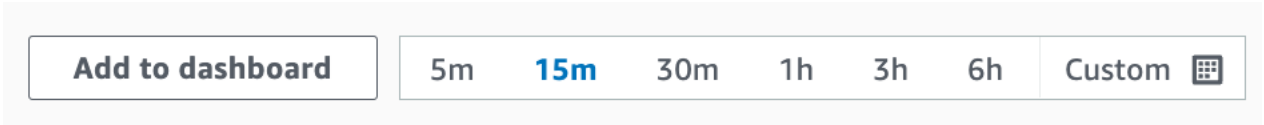
5. X-Ray SDK 會自動擷取經檢測的 AWS SDK 用戶端擲回的例外狀況，並記錄堆疊追蹤。



CloudWatch console

使用 CloudWatch 主控台

1. 開啟 CloudWatch 主控台的 X-Ray 追蹤映射頁面。
2. 主控台會顯示 X-Ray 從應用程式傳送的追蹤資料產生的服務圖表。請務必視需要調整追蹤映射的期間，以確保它在您第一次啟動 Web 應用程式後會顯示所有追蹤。



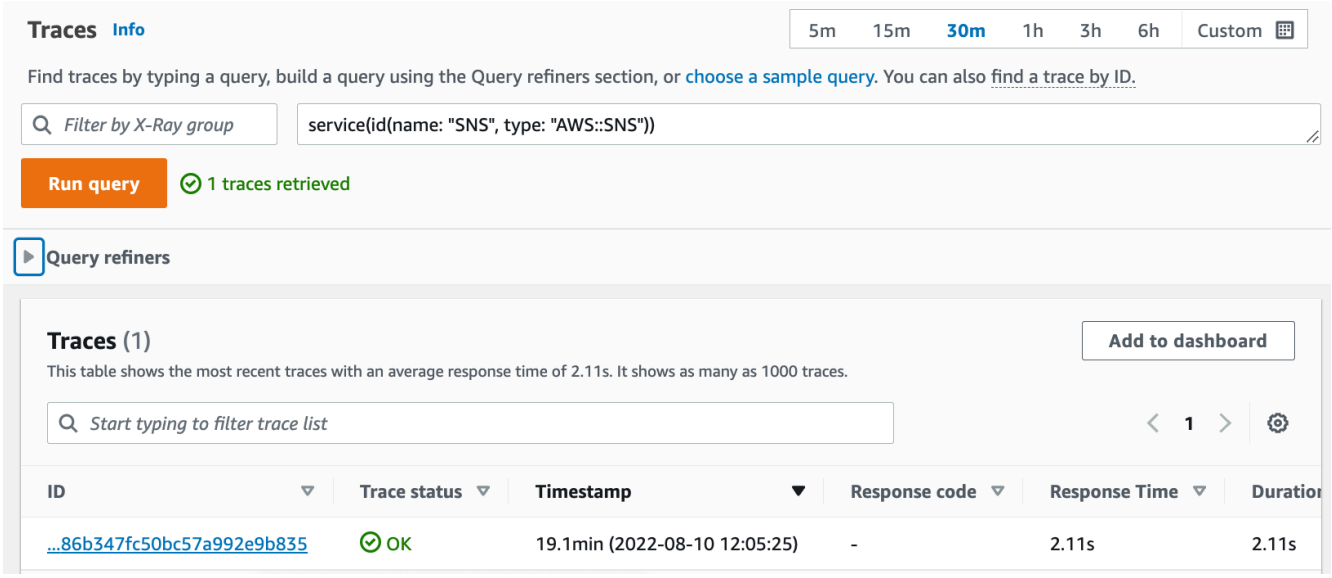
追蹤映射會顯示 Web 應用程式用戶端、在 Amazon EC2 中執行的 API，以及應用程式使用的每個 DynamoDB 資料表。每個向應用程式發出的請求都會在命中 API 時受到追蹤、產生向下游服務發出的請求並完成，其數量上限為可設定的每秒請求數量上限。

您可以在服務圖表中選擇任何節點來檢視產生流量至該節點的請求追蹤。目前，Amazon SNS 節點為橘色。向下切入來了解原因。

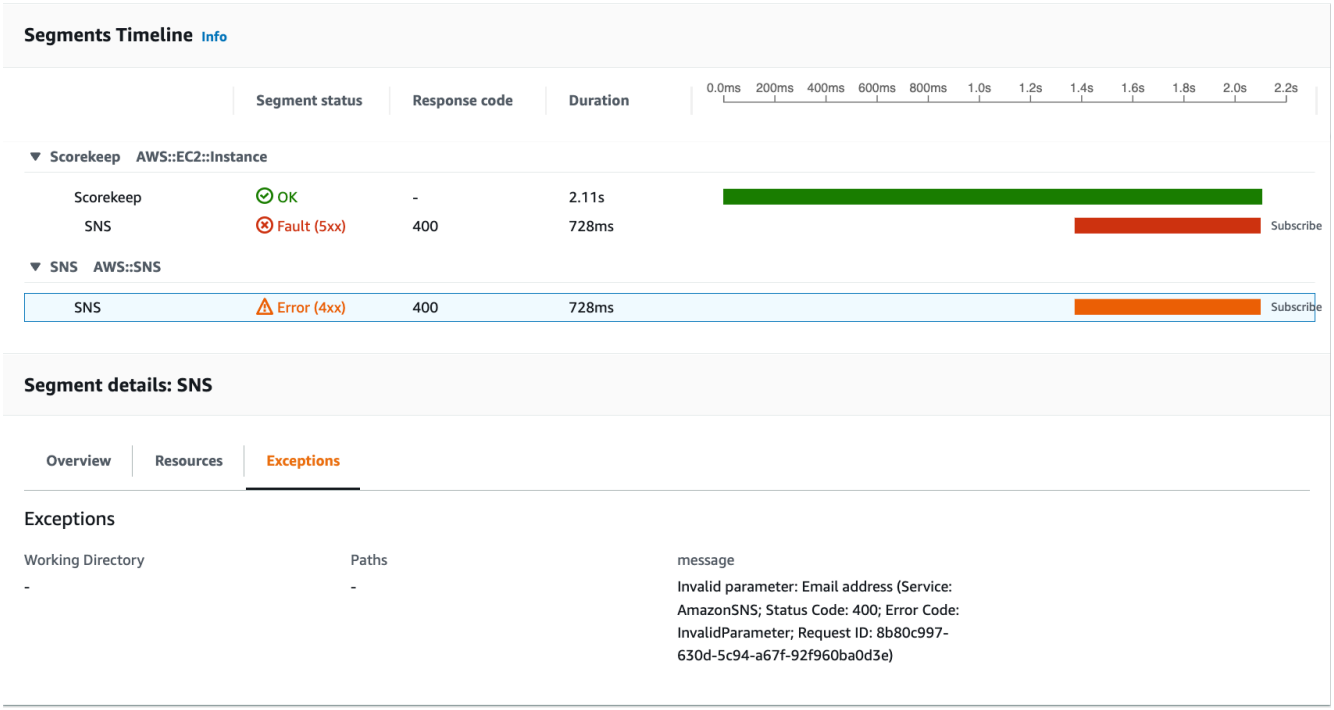

scorekeep-state
DynamoDB Table
scorekeep-game
DynamoDB Table
scorekeep-move
DynamoDB Table
Client
Scorekeep
ECS Container
SNS
SNS
scorekeep-session
DynamoDB Table

尋找錯誤原因

1. 選擇名為 SNS 的節點。SNS 節點詳細資訊面板會顯示在地圖下方。
2. 選擇檢視追蹤以存取追蹤頁面。
3. 新增頁面底部，從追蹤清單中選擇追蹤。此追蹤不具有方法或 URL，因為它是在啟動期間，而非回應傳入請求時記錄。



4. 選擇區段時間軸底部的 Amazon SNS 子區段，然後選擇 SNS 子區段的例外狀況索引標籤，以檢視例外狀況詳細資訊。



原因指出在 WebConfig 類別中對 createSubscription 發出的呼叫內所提供的電子郵件地址無效。在下一節中，我們將修正此問題。

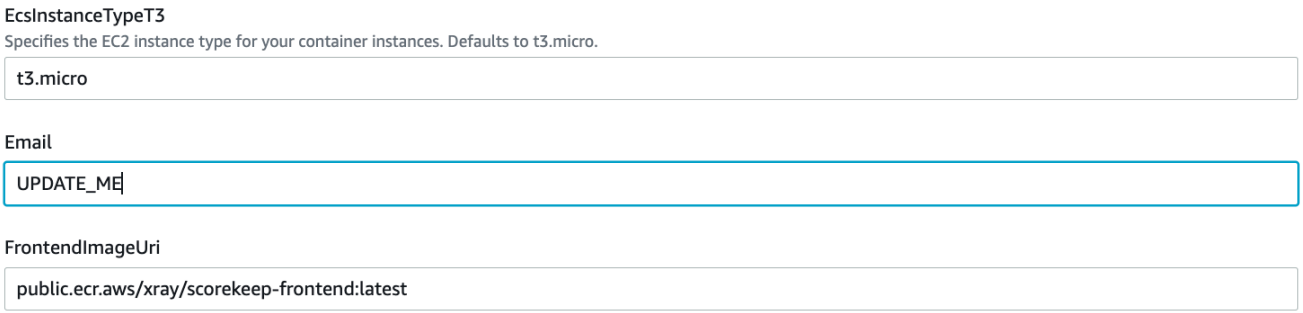
設定 Amazon SNS 通知

Scorekeep 使用 Amazon SNS，在使用者完成遊戲時傳送通知。當應用程式啟動時，它會嘗試為 CloudFormation 堆疊參數中定義的電子郵件地址建立訂閱。該呼叫目前失敗。設定通知電子郵件以啟用通知，並解決追蹤映射中反白顯示的失敗。

AWS Management Console

使用 設定 Amazon SNS 通知 AWS Management Console

1. 開啟 CloudFormation 主控台
2. 選擇清單中scorekeep堆疊名稱旁的選項按鈕，然後選擇更新。
3. 確定已選擇使用目前的範本，然後在更新堆疊頁面上按一下下一步。
4. 在清單中尋找電子郵件參數，並以有效的電子郵件地址取代預設值。



5. 向下捲動到頁面底部並選擇 Next (下一步)。
6. 捲動至檢閱頁面底部，選擇確認 CloudFormation 可能會使用自訂名稱建立 IAM 資源的核取方塊，然後選擇更新堆疊。
7. CloudFormation 堆疊現在正在更新。堆疊狀態將大約UPDATE_IN_PROGRESS為五分鐘，然後再變更為 UPDATE_COMPLETE。狀態會定期重新整理，或者您可以重新整理頁面。

AWS CLI

使用 設定 Amazon SNS 通知 AWS CLI

1. 導覽至您先前建立的xray-scorekeep/cloudformation/資料夾，然後在文字編輯器中開啟 cf-resources.yaml 檔案。
2. 尋找電子郵件參數中的Default值，並將其從 *UPDATE_ME* 變更為有效的電子郵件地址。

```
Parameters:
  Email:
    Type: String
    Default: UPDATE_ME # <- change to a valid abc@def.xyz email address
```

3. 從 `cloudformation` 資料夾，使用以下 AWS CLI 命令更新 CloudFormation 堆疊：

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

4. 等到 CloudFormation 堆疊狀態為 `UPDATE_COMPLETE`，這將需要幾分鐘的時間。使用下列 AWS CLI 命令來檢查狀態：

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

更新完成時，Scorekeep 會重新啟動並建立 SNS 主題訂閱。檢查您的電子郵件以確認訂閱，在您完成遊戲時查看更新。開啟追蹤映射，以確認對 SNS 的呼叫不再失敗。

探索範例應用程式

範例應用程式是 Java 中的 HTTP Web API，設定為使用適用於 Java 的 X-Ray 開發套件。當您使用 CloudFormation 範本部署應用程式時，它會建立在 ECS 上執行 Scorekeep 所需的 DynamoDB 資料表、Amazon ECS 叢集和其他服務。ECS 的任務定義檔案是透過 CloudFormation 建立。此檔案定義 ECS 叢集中每個任務使用的容器映像。這些影像是從官方 X-Ray 公有 ECR 取得。scorekeep API 容器映像使用 Gradle 編譯 API。Scorekeep 前端容器的容器映像會使用 nginx 代理伺服器為前端提供服務。此伺服器會將請求路由至以 /api 開頭的路徑，並傳送至 API。

為了檢測傳入 HTTP 請求，應用程式會新增軟體開發套件提供的 `TracingFilter`。

Example src/main/java/scorekeep/WebConfig.java - servlet 篩選條件

```
import javax.servlet.Filter;
import com.amazonaws.xray.javax.servlet.AWSXRayServletFilter;
...

@Configuration
public class WebConfig {

  @Bean
```

```
  public Filter TracingFilter() {
    return new AWSXRayServletFilter("Scorekeep");
  }
...
```

此篩選條件會傳送應用程式處理的所有傳入請求相關追蹤資料，包括請求 URL、方法、回應狀態、開始時間及結束時間。

應用程式也會使用 對 DynamoDB 進行下游呼叫 適用於 Java 的 AWS SDK。為了檢測這些呼叫，應用程式只會將 AWS SDK 相關的子模組視為相依性，而適用於 Java 的 X-Ray 開發套件會自動檢測所有 AWS SDK 用戶端。

應用程式使用 Docker建置執行個體上的原始程式碼，並使用 Gradle Docker Image和 Scorekeep API Dockerfile 檔案來執行 Gradle 在其 產生的可執行 JARENTRYPOINT。

Example 使用 Docker 透過 Gradle Docker Image 建置

```
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" ]
```

build.gradle 檔案會在編譯期間，透過將其宣告為依存項目，來從 Maven 下載軟體開發套件子模組。

Example build.gradle -- 相依性

```
...
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 AWS SDK Instrumentor 子模組都是自動檢測使用 AWS SDK 進行的任何下游呼叫所需的。

若要將原始區段資料轉送至 X-Ray API，X-Ray 協助程式需要接聽 UDP 連接埠 2000 上的流量。若要這樣做，應用程式會在與 ECS 上的 Scorekeep 應用程式一起部署的容器中執行 X-Ray 協助程式，做為附屬容器。如需詳細資訊，請參閱 X-Ray 協助程式主題。

Example ECS 任務定義中的 X-Ray 協助程式容器定義

```
...
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
      ...
```

適用於 Java 的 X-Ray 開發套件提供名為 的類別`AWSXRay`，提供全域記錄器，您可以用來檢測程式碼`TracingHandler`的 。您可以設定全域記錄器來自訂為傳入 HTTP 呼叫建立區段的 `AWSXRayServletFilter`。範例包括 `WebConfig` 類別中的靜態區塊，該區塊會使用外掛程式和抽樣規則設定全域記錄器。

Example src/main/java/scorekeep/WebConfig.java - 記錄器

```
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());
    ...

  }
}
```

此範例使用建立器從名為 sampling-rules.json 的檔案載入抽樣規則。抽樣規則會判斷軟體開發套件記錄傳入請求區段的速率。

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
  }
}
```

抽樣規則檔案會定義四個自訂抽樣規則及預設規則。針對每個傳入請求，軟體開發套件會依照定義規則的順序評估自訂規則。軟體開發套件會套用第一個與請求方法、路徑及服務名稱相符的規則。針對 Scorekeep，第一個規則會透過以 1.0 的速率每秒套用一個請求的單一固定目標，或是在滿足固定目標之後套用 100% 的請求，來追補所有 POST 請求 (資源建立呼叫)。

其他三個自訂規則會在沒有指向工作階段、遊戲和狀態讀取 (GET 請求) 固定目標的情況下套用 5% 的速率 這會將前端為了確保內容處於最新狀態，每幾秒鐘自動發出定期呼叫的追蹤數量降至最低。針對所有其他請求，檔案會定義每秒單一請求的預設速率及 10% 的速率。

範例應用程式也會示範如何使用進階功能 (例如手動軟體開發套件用戶端檢測、建立額外子區段及傳出 HTTP 呼叫)。如需詳細資訊，請參閱AWS X-Ray 範例應用程式。

選用：最低權限政策

Scorekeep ECS 容器使用完整存取政策存取資源，例如 `AmazonSNSFullAccess`和 `AmazonDynamoDBFullAccess`。使用完整存取政策不是生產應用程式的最佳實務。下列範例會更新

DynamoDB IAM 政策，以改善應用程式的安全性。若要進一步了解 IAM 政策中的安全最佳實務，請參閱 AWS X-Ray 的身分和存取管理。

Example cf-resources.yaml 範本 ECSTaskRole 定義

```
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"
```

若要更新您的政策，請先識別 DynamoDB 資源的 ARN。然後，您可以在自訂 IAM 政策中使用 ARN。最後，您將該政策套用到您的執行個體描述檔。

若要識別 DynamoDB 資源的 ARN：

1. 開啟 DynamoDB 主控台。
2. 從左側導覽列選擇資料表。
3. 選擇任一 `scorekeep-*`以顯示資料表詳細資訊頁面。
4. 在概觀索引標籤下，選擇其他資訊以展開區段，並檢視 Amazon Resource Name (ARN)。複製這個值。
5. 將 ARN 插入下列 IAM 政策，將 `AWS_REGION`和 `AWS_ACCOUNT_ID`值取代為您的特定區域和帳戶 ID。此新政策僅允許指定的動作，而非允許任何動作`AmazonDynamoDBFullAccess`的政策。

   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-*"
        }
    ]
}
```

應用程式建立的資料表遵循一致的命名慣例。您可以使用 `scorekeep-*` 格式來指示所有 Scorekeep 資料表。

變更您的 IAM 政策

1. 從 IAM 主控台開啟 Scorekeep 任務角色 (scorekeepRole)。
2. 選擇AmazonDynamoDBFullAccess政策旁的核取方塊，然後選擇移除以移除此政策。
3. 選擇新增許可，然後選擇連接政策，最後選擇建立政策。
4. 選擇 JSON 索引標籤，並貼入上面建立的政策。
5. 選擇頁面底部的下一步：標籤。
6. 選擇頁面底部的下一步：檢閱。
7. 針對名稱，指派政策的名稱。
8. 選擇頁面底部的建立政策。
9. 將新建立的政策連接至scorekeepRole角色。連接的政策可能需要幾分鐘的時間才會生效。

如果您已將新政策連接至scorekeepRole角色，則必須先將其分離，才能刪除 CloudFormation 堆疊，因為此連接政策會封鎖堆疊遭到刪除。刪除政策可自動分離政策。

移除您的自訂 IAM 政策

1. 開啟 IAM 主控台。
2. 從左側導覽列選擇政策。
3. 搜尋您稍早在本節中建立的自訂政策名稱，然後選擇政策名稱旁的選項按鈕以反白顯示。
4. 選擇動作下拉式清單，然後選擇刪除。
5. 輸入自訂政策的名稱，然後選擇刪除以確認刪除。這會自動分離政策與`scorekeepRole`角色。

清除

請依照下列步驟刪除 Scorekeep 應用程式資源：

> **Note**
>
> 如果您使用本教學課程的上一節建立並連接自訂政策，您必須先從 移除政策，`scorekeepRole`才能刪除 CloudFormation 堆疊。

AWS Management Console

使用 刪除範例應用程式 AWS Management Console

1. 開啟 CloudFormation 主控台
2. 選擇清單中`scorekeep`堆疊名稱旁的選項按鈕，然後選擇刪除。
3. CloudFormation 堆疊現在正在刪除。堆疊狀態將`DELETE_IN_PROGRESS`保持幾分鐘，直到刪除所有資源為止。狀態會定期重新整理，或者您可以重新整理頁面。

AWS CLI

使用 刪除範例應用程式 AWS CLI

1. 輸入下列 AWS CLI 命令來刪除 CloudFormation 堆疊：

```
aws cloudformation delete-stack --stack-name scorekeep
```

2. 等到 CloudFormation 堆疊不再存在，這大約需要五分鐘的時間。使用下列 AWS CLI 命令來檢查狀態：

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

後續步驟

在下一章 中進一步了解 X-RayAWS X-Ray 概念。

若要檢測您自己的應用程式，請進一步了解適用於 Java 的 X-Ray 開發套件或其他其中一個 X-Ray SDKs：

- 適用於 Java 的 X-Ray 開發套件 – AWS X-Ray 適用於 Java 的 SDK
- 適用於 Node.js 的 X-Ray 開發套件 – AWS 適用於 Node.js 的 X-Ray 開發套件
- 適用於 .NET 的 X-Ray 開發套件 – AWS X-Ray 適用於 .NET 的 SDK

若要在本機或 上執行 X-Ray 協助程式 AWS，請參閱 AWS X-Ray 協助程式。

若要在 GitHub 上參與範例應用程式，請參閱 eb-java-scorekeep。

手動檢測 AWS SDK 用戶端

當您在建置相依性中包含 AWS SDK Instrumentor 子模組時，適用於 Java 的 X-Ray 開發套件會自動檢測所有 SDK 用戶端。 AWS

若要停用自動用戶端檢測，您可以移除 Instrumentor 子模組。這可讓您手動檢測一些特定用戶端而忽略其他用戶端，或使用不同用戶端上的不同追蹤處理常式。

為了說明支援檢測特定 AWS SDK 用戶端，應用程式會將追蹤處理常式傳遞至 ，`AmazonDynamoDBClientBuilder`做為使用者、遊戲和工作階段模型中的請求處理常式。此程式碼變更會通知 SDK 使用這些用戶端來檢測對 DynamoDB 的所有呼叫。

Example **`src/main/java/scorekeep/SessionModel.java`** – 手動 AWS SDK 用戶端檢測

```
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);
```

如果您從專案相依性中移除 AWS SDK Instrumentor 子模組，則只有手動檢測的 AWS SDK 用戶端會出現在追蹤映射中。

建立其他子區段

在使用者模型類別中，應用程式會手動建立子區段以分組 `saveUser` 函數內發出的所有下游呼叫，並新增中繼資料。

Example **src/main/java/scorekeep/UserModel.java** - 自訂子區段

```
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();
    }
  }
```

記錄標註、中繼資料及使用者 ID

在遊戲模型類別中，每次在 DynamoDB 中儲存遊戲時，應用程式都會記錄中繼資料區塊中的Game物件。應用程式還會另外在標註中記錄遊戲 ID，用於篩選條件表達式。

Example **src/main/java/scorekeep/GameModel.java** – 註釋和中繼資料

```
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();
    }
  }
```

在移動控制器中，應用程式會使用 記錄使用者 IDssetUser。區段會將使用者 ID 記錄在單獨的欄位中，並建立索引以用於搜尋。

Example src/main/java/scorekeep/MoveController.java – 使用者 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);
  }
```

檢測傳出的 HTTP 呼叫

使用者工廠類別顯示應用程式如何使用適用於 Java 的 X-Ray 開發套件版本 HTTPClientBuilder 來檢測傳出的 HTTP 呼叫。

Example **src/main/java/scorekeep/UserFactory.java** – HTTPClient 檢測

```
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();
    }
  }
```

若您目前使用 `org.apache.http.impl.client.HttpClientBuilder`，您可以直接使用一個用於 `com.amazonaws.xray.proxies.apache.http.HttpClientBuilder` 的項目將該類別的匯入陳述式切換出去。

檢測 PostgreSQL 資料庫的呼叫

`application-pgsql.properties` 檔案會將 X-Ray PostgreSQL 追蹤攔截器新增至 中建立的資料來源RdsWebConfig.java。

Example **application-pgsql.properties** – PostgreSQL 資料庫檢測

```
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**
>
> 如需如何將 PostgreSQL 資料庫新增至應用程式環境的詳細資訊，請參閱《 AWS Elastic Beanstalk 開發人員指南》中的使用 Elastic Beanstalk 設定資料庫。

`xray` 分支中的 X-Ray 示範頁面包含示範，該示範使用檢測的資料來源來產生追蹤，以顯示其產生的 SQL 查詢的相關資訊。導覽至執行中應用程式的 `/#/xray` 路徑，或選擇導覽列中的 Powered by AWS X-Ray(採用 &xraylong; 技術)，以查看示範頁面。

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.

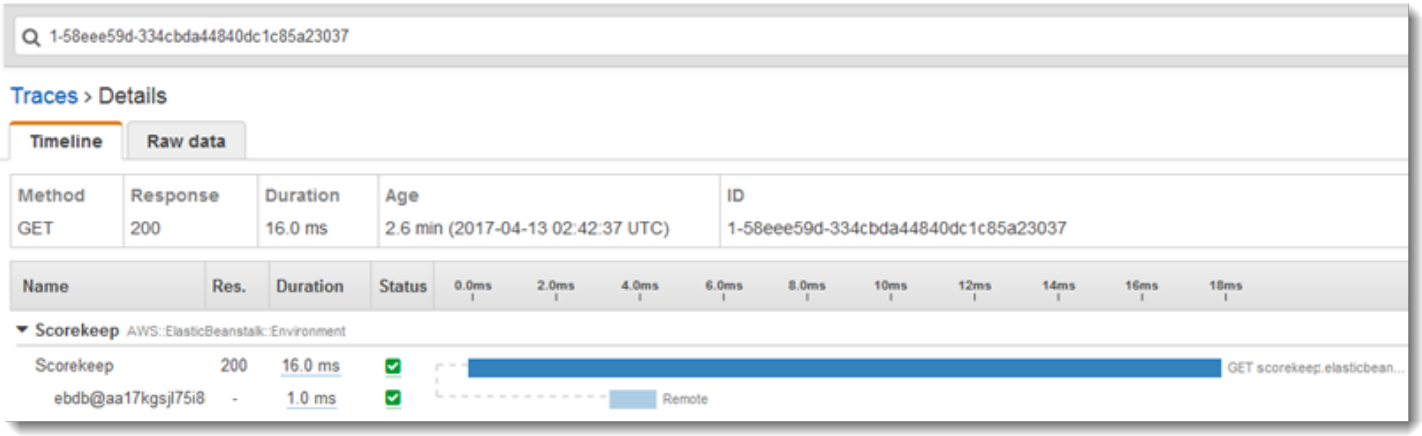
Trace SQL queries

View traces in AWS X-Ray

ID	Winner	Loser
1	Mugur	Gheorghiță
2	Paula	Adorján
3	Αρχίας	Stela
4	付	Pərvanə

選擇 Trace SQL queries (追蹤 SQL 查詢) 以模擬遊戲工作階段，並將結果存放在連接的資料庫中。然後，選擇在 AWS X-Ray 中檢視追蹤，以查看命中 API `/api/history`路由的已篩選追蹤清單。

從清單中選擇其中一個追蹤以查看時間軸，包括 SQL 查詢。



檢測 AWS Lambda 函數

Scorekeep 使用兩個 AWS Lambda 函數。第一個是來自 `lambda` 分支的 Node.js 函數，會為新使用者產生隨機名稱。當使用者在未輸入名稱的情況下建立工作階段時，應用程式會使用 適用於 Java 的 AWS SDK呼叫名為 `random-name` 的函數。適用於 Java 的 X-Ray 開發套件會在子區段中記錄對 Lambda 呼叫的相關資訊，就像使用經檢測的 AWS 開發套件用戶端進行的任何其他呼叫一樣。

> **Note**
>
> 執行 `random-name` Lambda 函數需要在 Elastic Beanstalk 環境之外建立其他資源。如需詳細資訊和指示，請參閱讀我檔案：AWS Lambda 整合。

第二個函數 (`scorekeep-worker`) 是一種 Python 函數，會在 Scorekeep API 之外獨立執行。遊戲結束時，API 會將工作階段 ID 及遊戲 ID 寫入 SQS 佇列。工作者函數會從佇列讀取項目，並呼叫 Scorekeep API 來建構每個遊戲工作階段的完整記錄，以便在 Amazon S3 中儲存。

Scorekeep 包含 AWS CloudFormation 範本和指令碼來建立這兩個函數。由於您需要將 X-Ray 開發套件與函數程式碼綁定，因此範本會建立不含任何程式碼的函數。部署 Scorekeep 時，包含在 `.ebextensions` 資料夾中的組態檔會建立包含軟體開發套件的來源套件，並使用 AWS Command Line Interface更新函數程式碼及組態。

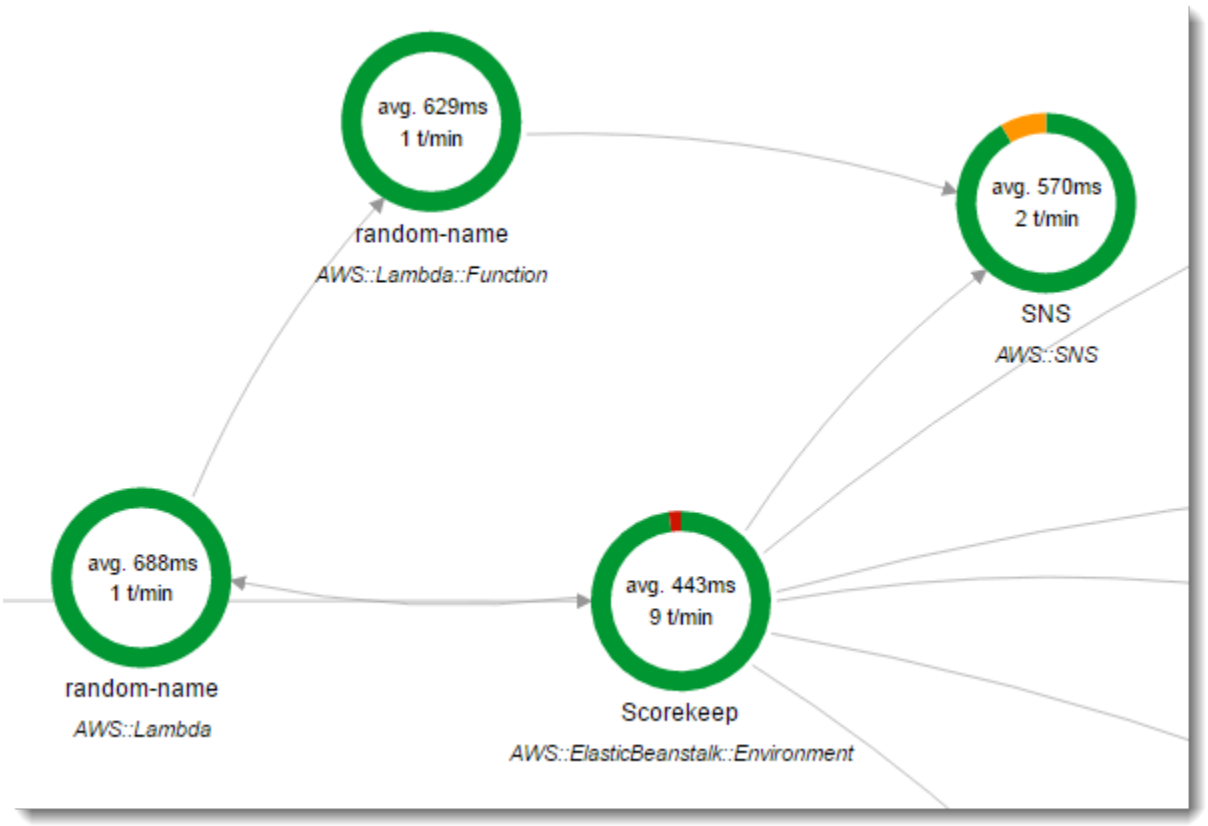
函數

隨機名稱

Scorekeep 會在使用者未登入或未指定任何使用者名稱啟動遊戲工作階段時呼叫隨機名稱函數。當 Lambda 處理對 的呼叫時`random-name`，它會讀取追蹤標頭，其中包含由適用於 Java 的 X-Ray 開發套件所撰寫的追蹤 ID 和取樣決策。

對於每個抽樣請求，Lambda 會執行 X-Ray 協助程式並寫入兩個區段。第一個區段會記錄叫用函數的 Lambda 呼叫相關資訊。此區段包含與 Scorekeep 記錄的子區段相同的資訊，但從 Lambda 觀點來看。第二個區段則代表函數進行的工作。

Lambda 透過函數內容將函數區段傳遞至 X-Ray SDK。當您檢測 Lambda 函數時，不會使用 SDK 為傳入請求建立區段。Lambda 提供區段，您可以使用 SDK 來檢測用戶端和寫入子區段。



`random-name` 函數實作於 Node.js 中。它使用 Node.js 中適用於 JavaScript 的 SDK 來傳送 Amazon SNS 的通知，並使用適用於 Node.js 的 X-Ray 開發套件來檢測 AWS SDK 用戶端。為寫入標註，函數

會使用 AWSXRay.captureFunc 建立子區段，然後在受檢測函數中寫入標註。在 Lambda 中，您無法將註釋直接寫入函數區段，只能寫入您建立的子區段。

Example **function/index.js** -- 隨機名稱 Lambda 函數

```
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;
```

此函數會在您將簡易應用程式部署到 Elastic Beanstalk 時自動建立。`xray` 分支包含用來建立空白 Lambda 函數的指令碼。`.ebextensions` 資料夾中的組態檔案會在部署`npm install`期間使用 建置函數套件，然後使用 CLI 更新 Lambda AWS 函數。

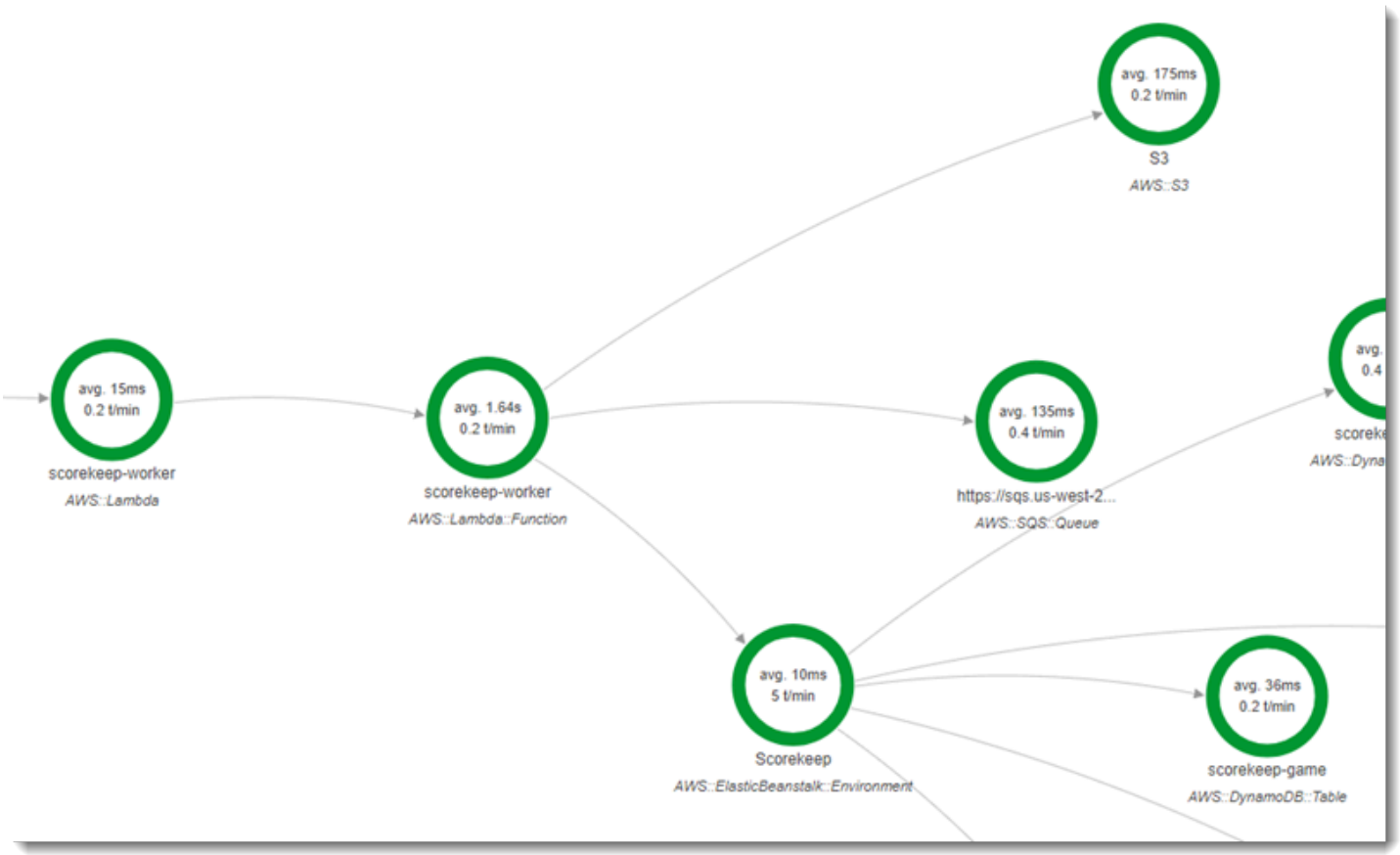
工作程序

受檢測工作者函數會在自己的分支 (`xray-worker`) 中提供，因為除非您先建立工作者函數及相關資源，否則它便無法執行。如需說明，請參閱分支讀我檔案。

函數由綁定的 Amazon CloudWatch Events 事件每 5 分鐘觸發一次。執行時，函數會從 Scorekeep 管理的 Amazon SQS 佇列中提取項目。每個訊息都包含完整遊戲的相關資訊。

工作者會從遊戲記錄參考的其他資料表提取遊戲記錄及文件。例如，DynamoDB 中的遊戲記錄包含遊戲期間執行的移動清單。清單不包含移動本身，但包含存放在單獨資料表中移動的 ID。

工作階段及狀態也會以參考存放。這可避免遊戲資料表中的項目過大，但需要額外的呼叫才能取得所有遊戲相關資訊。工作者會解譯所有這些項目，並在 Amazon S3 中建構遊戲的完整記錄做為單一文件。當您想要對資料進行分析時，可以使用 Amazon Athena 直接在 Amazon S3 中對其執行查詢，而無需執行大量讀取資料遷移，以從 DynamoDB 中獲取資料。 Amazon Athena



工作者函數在其位於 AWS Lambda內的組態中已啟用主動式追蹤。與隨機名稱函數不同，工作者不會收到來自經檢測應用程式的請求，因此 AWS Lambda 不會收到追蹤標頭。透過主動追蹤，Lambda 會建立追蹤 ID 並做出抽樣決策。

適用於 Python 的 X-Ray 開發套件只是函數頂端的幾行，可匯入開發套件並執行其`patch_all`函數來修補用於呼叫 Amazon SQS AWS SDK for Python (Boto) 和 Amazon S3 的 和 HTTclients。工作者呼叫 Scorekeep API 時，軟體開發套件會將追蹤標頭新增至請求，來追蹤透過 API 進行的呼叫。

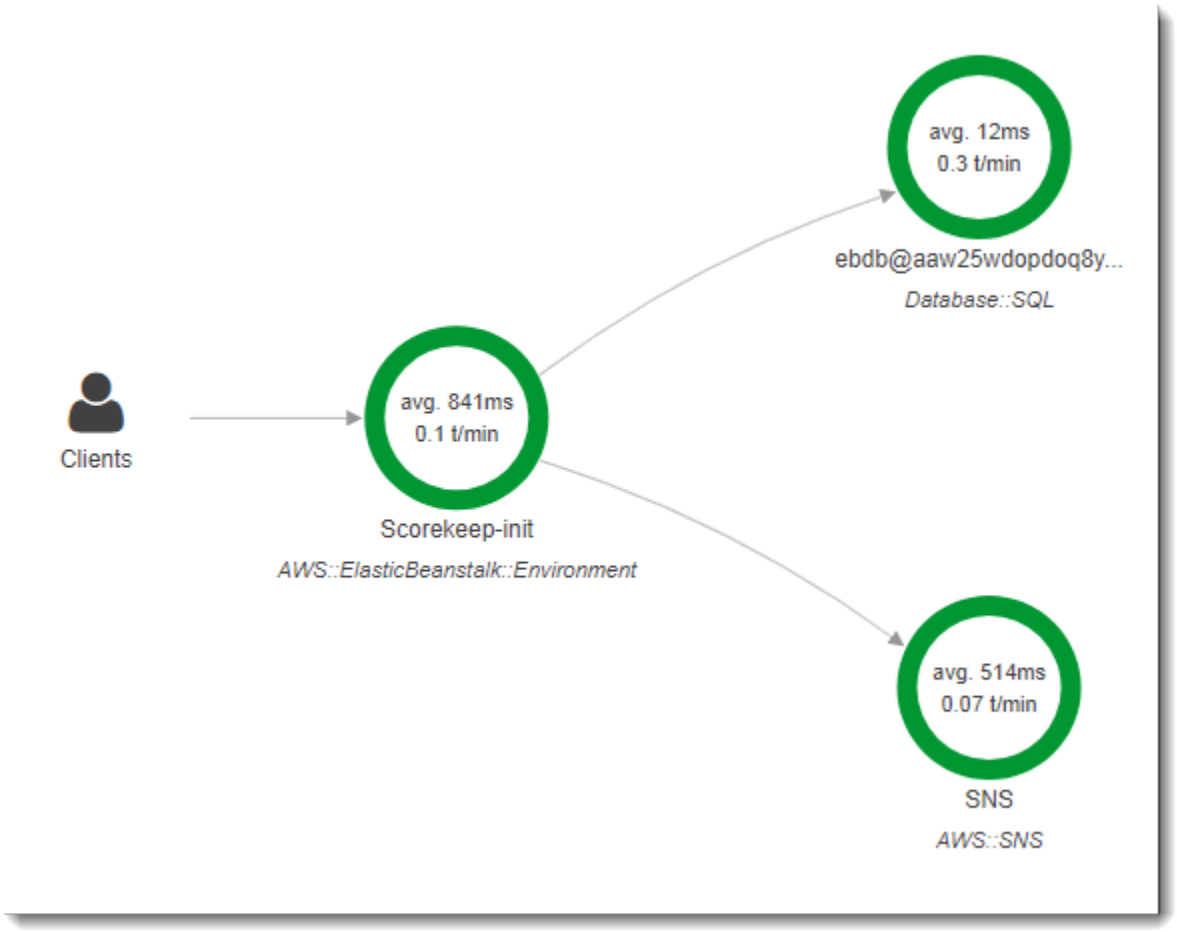
Example **_lambda/scorekeep-worker/scorekeep-worker.py** -- 工作者 Lambda 函數

```
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
    )
   ...
```

檢測啟動程式碼

適用於 Java 的 X-Ray 開發套件會自動為傳入請求建立區段。只要請求是在範圍內，您即可使用經檢測的用戶端並記錄子區段，而不會出現問題。如果您嘗試在啟動程式碼中使用經檢測的用戶端，則會收到 SegmentNotFoundException。

啟動程式碼會在 Web 應用程式的標準請求/回應流程之外執行，因此您需要手動建立區段以檢測起始程式碼。Scorekeep 會在其 `WebConfig` 檔案中顯示啟動程式碼的檢測。Scorekeep 會在啟動期間呼叫 SQL 資料庫和 Amazon SNS。



預設`WebConfig`類別會為通知建立 Amazon SNS 訂閱。若要在使用 Amazon SNS 用戶端時提供 X-Ray 開發套件寫入的區段，Scorekeep 會在全域記錄器`endSegment`上呼叫 `beginSegment`和 。

Example **`src/main/java/scorekeep/WebConfig.java`** – 啟動程式碼中的檢測 AWS SDK 用戶端

```
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 RdsWebConfigScorekeep 在連接 Amazon RDS 資料庫時使用的 中，組態也會為 SQL 用戶端建立區段，讓休眠在啟動期間套用資料庫結構描述時使用該區段。

Example **src/main/java/scorekeep/RdsWebConfig.java** – 在啟動程式碼中檢測 SQL 資料庫用戶端

```
@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 會自動執行，並使用 SQL 用戶端。由於用戶端已經過檢測，因此 Scorekeep 必須覆寫預設的實作，並提供呼叫用戶端時軟體開發套件使用的區段。

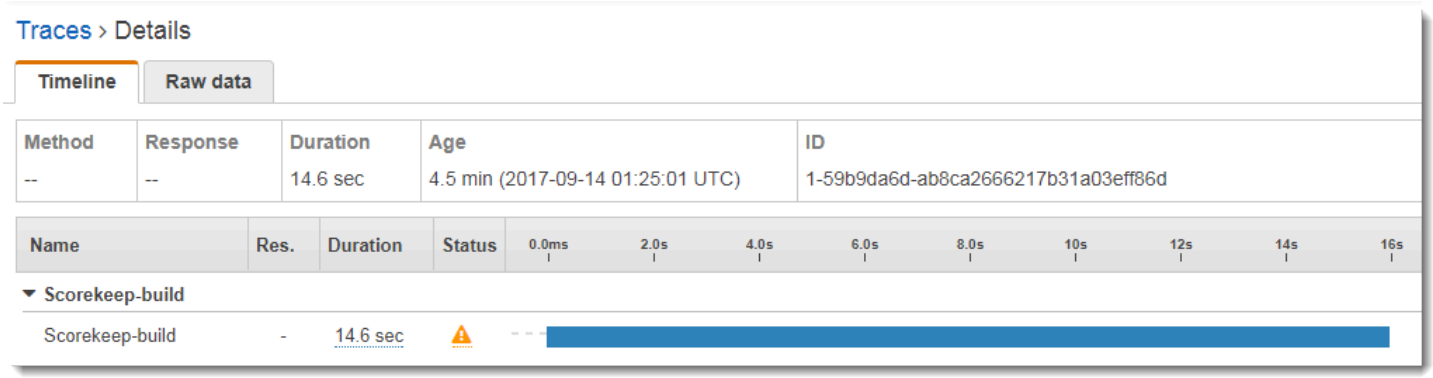
檢測指令碼

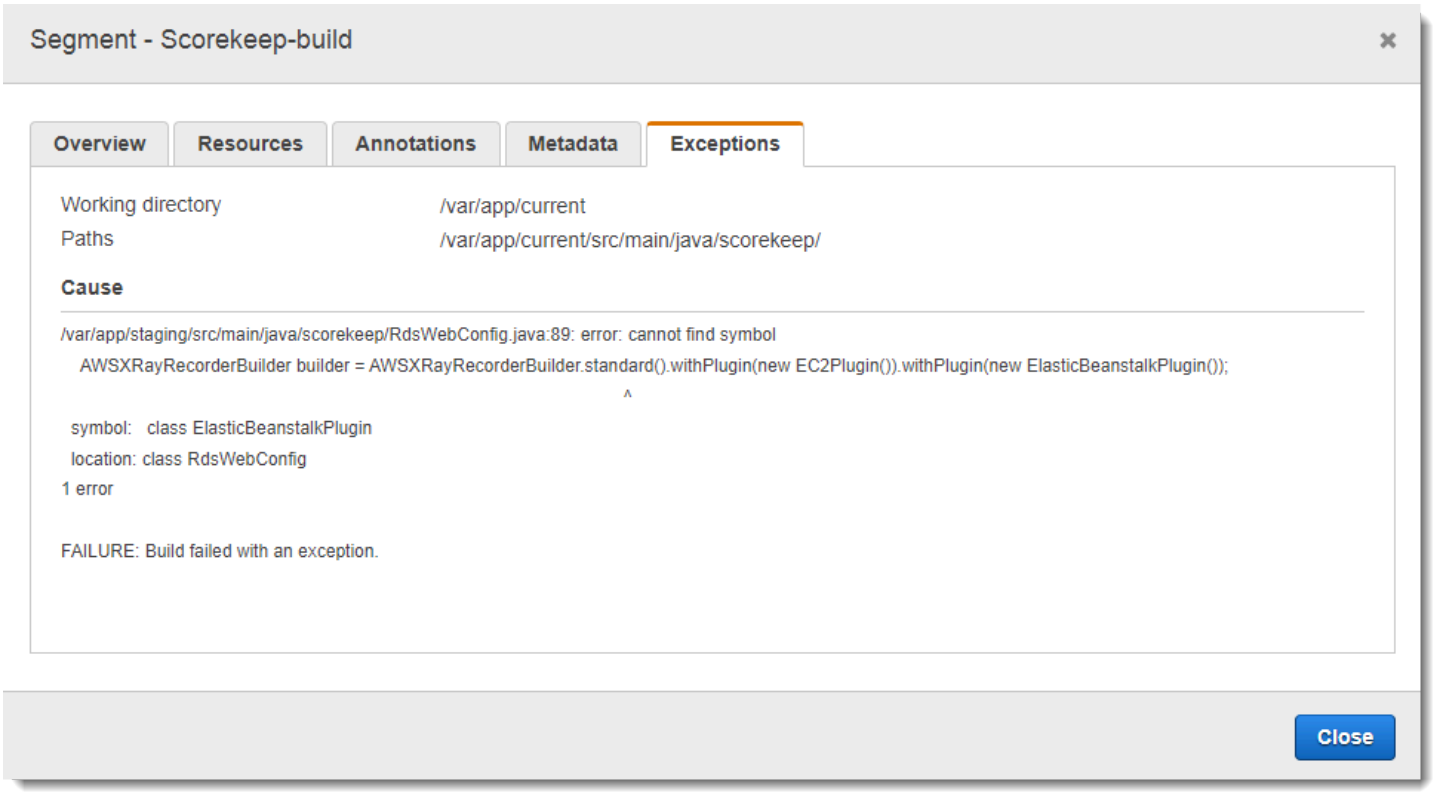
您也可以檢測不屬於您應用程式部分的程式碼。當 X-Ray 協助程式正在執行時，它會將收到的任何區段轉送至 X-Ray，即使它們不是由 X-Ray SDK 產生。Scorekeep 會使用自己的指令碼檢測在部署期間編譯應用程式的建置。

Example **bin/build.sh** – 經檢測的建置指令碼

```
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 是簡易的 Python 指令碼，可建構區段物件，將它們轉換成 JSON 文件，並透過 UDP 傳送到精靈。如果 Gradle 建置失敗，您可以在 X-Ray 主控台追蹤映射中按一下 scorekeep-build 節點，以尋找錯誤訊息。





檢測 Web 應用程式用戶端

在xray-cognito分支中，Scorekeep 使用 Amazon Cognito 讓使用者能夠建立 帳戶，並使用該帳戶登入，從 Amazon Cognito 使用者集區擷取使用者資訊。當使用者登入時，Scorekeep 會使用 Amazon Cognito 身分集區來取得臨時 AWS 登入資料，以便與 搭配使用 適用於 JavaScript 的 AWS SDK。

Identity Pool 是設定為可讓登入的使用者將追蹤資料寫入 AWS X-Ray。Web 應用程式會使用這些登入資料來記錄登入使用者的 ID、瀏覽器路徑和用戶端對 Scorekeep API 的呼叫檢視。

大部分工作都會在名稱為 `xray` 的服務類別中完成。此服務類別提供產生必要識別符、建立進行中區段、完成區段，以及將區段文件傳送至 X-Ray API 的方法。

Example **public/xray.js** – 記錄和上傳客群

```
...
  service.beginSegment = function() {
    var segment = {};
    var traceId = '1-' + service.getHexTime() + '-' + service.getHexId(24);

    var id = service.getHexId(16);
    var startTime = service.getEpochTime();
```

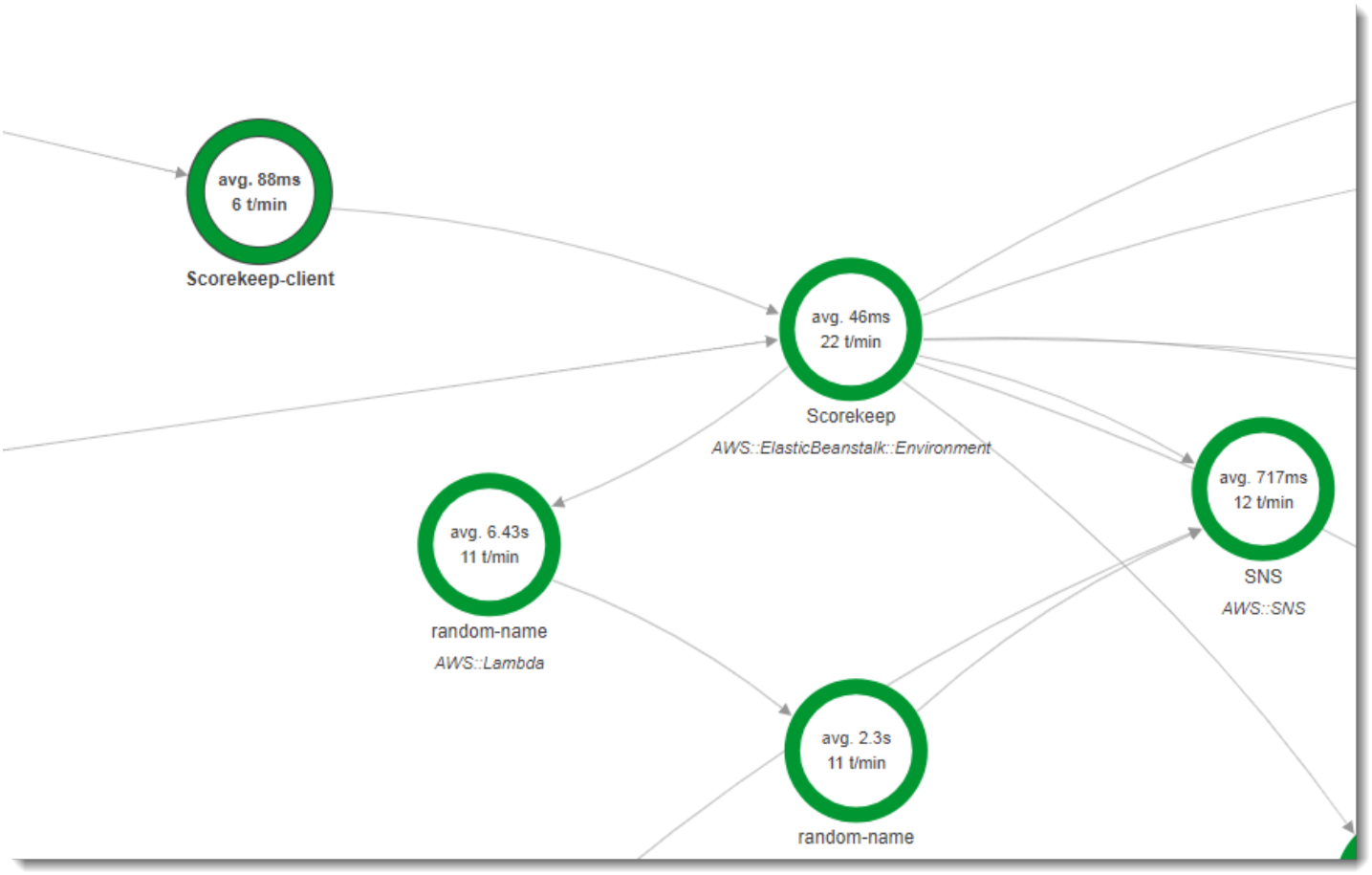
```
  segment.trace_id = traceId;
  segment.id = id;
  segment.start_time = startTime;
  segment.name = 'Scorekeep-client';
  segment.in_progress = true;
  segment.user =  sessionStorage['userid'];
  segment.http = {
    request: {
      url: window.location.href
    }
  };

  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);
    }
  })
}
```

您可在資源服務的標頭和 `transformResponse` 函數中呼叫這些方法，而 Web 應用程式會使用這些資源服務來呼叫 Scorekeep API。若要在與 API 產生的區段相同的追蹤中包含用戶端區段，Web 應用程式必須在 X-Ray SDK 可以讀取的追蹤標頭 (`X-Amzn-Trace-Id`) 中包含追蹤 ID 和區段 ID。當受檢測的 Java 應用程式收到具有此標頭的請求時，適用於 Java 的 X-Ray 開發套件會使用相同的追蹤 ID，並讓來自 Web 應用程式用戶端的區段成為其區段的父系。

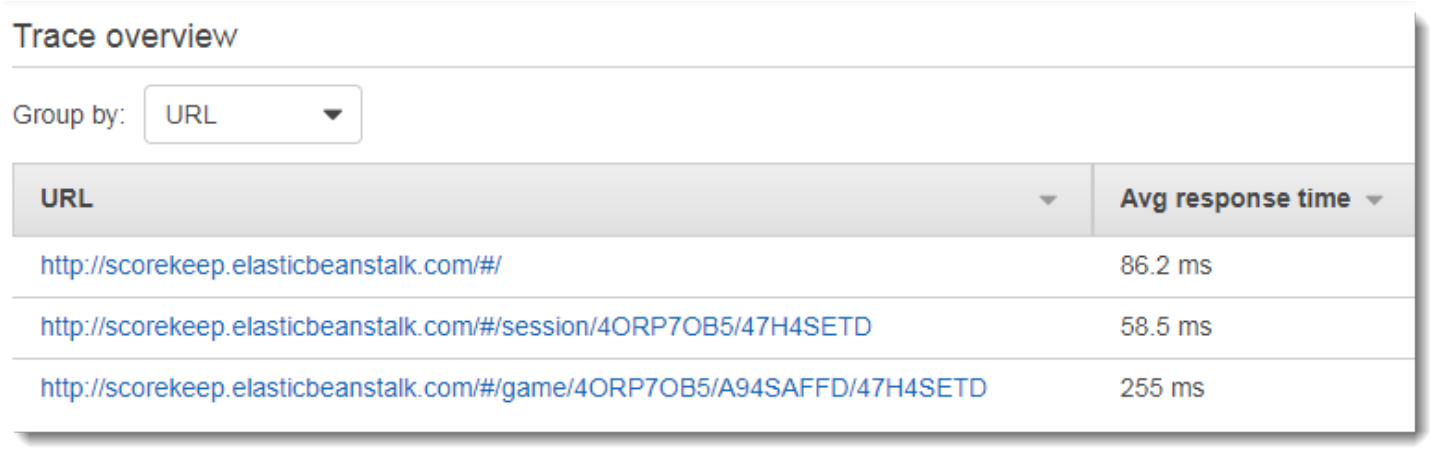
Example **public/app/services.js** – 記錄角度資源呼叫和寫入追蹤標頭的區段

```
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);
      },
    },
...
```

產生的追蹤映射包含 Web 應用程式用戶端的節點。



如果追蹤包括 Web 應用程式的區段，即會顯示使用者在瀏覽器中看到的 URL (路徑開頭為 `/#/`)。未使用用戶端檢測時，您只會取得 Web 應用程式呼叫之 API 資源的 URL (路徑開頭為 `/api/`)。



在工作者執行緒中使用受檢測用戶端

Scorekeep 使用工作者執行緒，在使用者贏得遊戲時將通知發佈至 Amazon SNS。發佈通知所花費的時間比其餘合併的請求操作更長，但不會影響用戶端或使用者。因此，若要改善回應時間，以非同步方式執行任務是一種好方法。

不過，適用於 Java 的 X-Ray 開發套件不知道在建立執行緒時哪個區段處於作用中狀態。因此，當您嘗試在執行緒中使用經檢測的 適用於 Java 的 AWS SDK 用戶端時，它會擲回 `SegmentNotFoundException`，使執行緒當機。

Example web-1.error.log

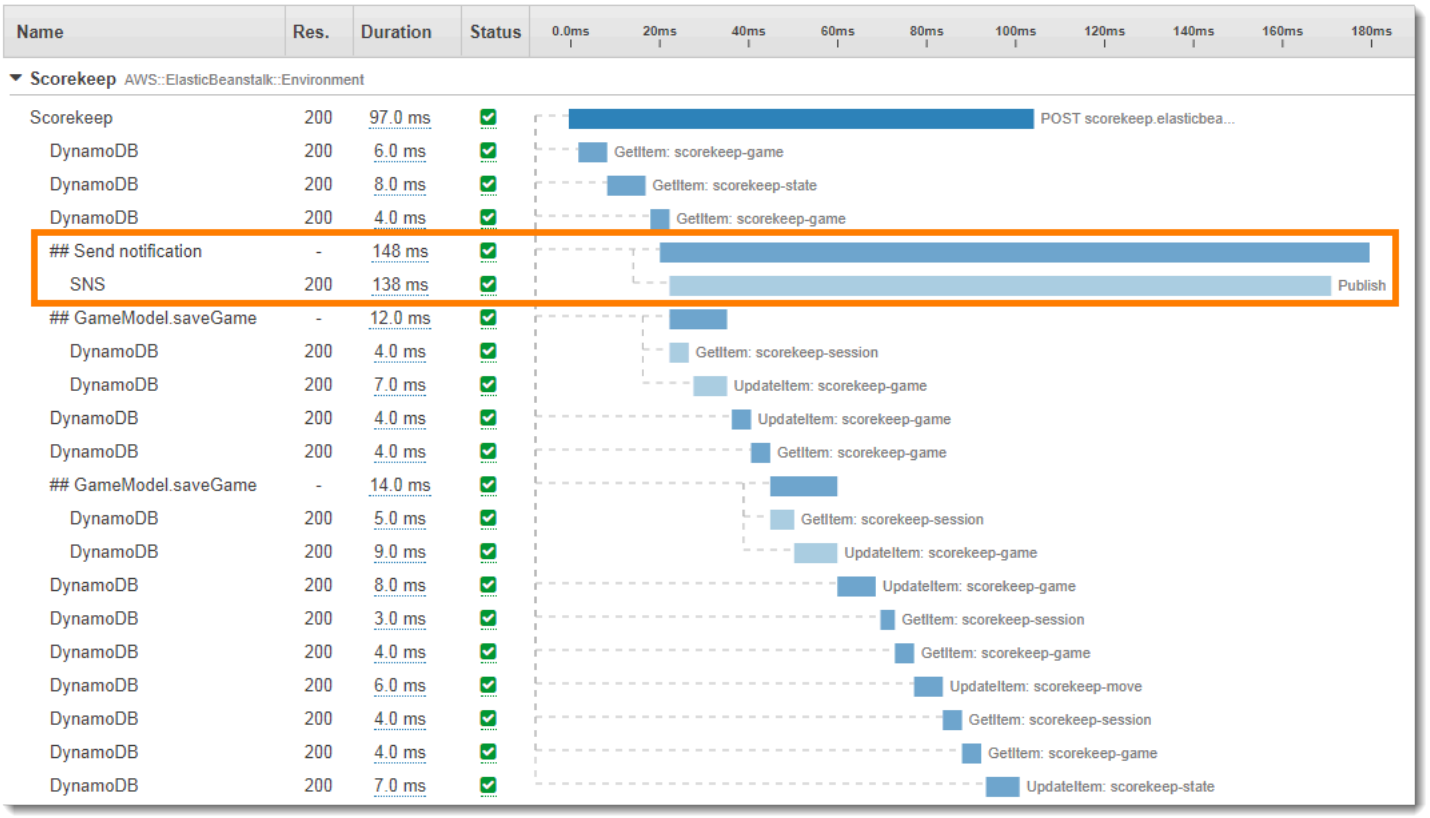
```
Exception in thread "Thread-2" com.amazonaws.xray.exceptions.SegmentNotFoundException:
 Failed to begin subsegment named 'AmazonSNS': segment cannot be found.
        at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
        at
 sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:62)
        at
 sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.ja
...
```

為了修正此問題，應用程式會使用 `GetTraceEntity`取得主執行緒中區段的參考，並`Entity.run()`安全地執行工作者執行緒程式碼，以存取區段的內容。

Example **src/main/java/scorekeep/MoveFactory.java** – 將追蹤內容傳遞至工作者執行緒

```
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();
          }
        }
```

由於請求現在會在呼叫 Amazon SNS 之前解決，因此應用程式會為執行緒建立單獨的子區段。這可防止 X-Ray 開發套件在記錄來自 Amazon SNS 的回應之前關閉區段。如果 Scorekeep 解決請求時未開啟任何子區段，Amazon SNS 的回應可能會遺失。



如需多執行緒的詳細資訊，請參閱[在多執行緒應用程式之間傳遞區段內容]。

AWS X-Ray 協助程式

> **Note**
>
> 您現在可以使用 CloudWatch 代理程式從 Amazon EC2 執行個體和內部部署伺服器收集指標、日誌和追蹤。CloudWatch 代理程式 1.300025.0 版及更新版本可以從 OpenTelemetry 或 X-Ray 用戶端 SDKs 收集追蹤，並將其傳送至 X-Ray。使用 CloudWatch 代理程式而非 AWS Distro for OpenTelemetry (ADOT) Collector 或 X-Ray 協助程式收集追蹤，可協助您減少管理的代理程式數量。如需詳細資訊，請參閱《CloudWatch 使用者指南》中的 CloudWatch 代理程式主題。 CloudWatch

AWS X-Ray 協助程式是一種軟體應用程式，可接聽 UDP 連接埠 2000 上的流量、收集原始區段資料，並將其轉送至 AWS X-Ray API。協助程式可搭配 AWS X-Ray SDKs 運作，且必須執行，以便 SDKs 傳送的資料可以到達 X-Ray 服務。X-Ray 協助程式是開放原始碼專案。您可以在 GitHub 上關注專案並提交問題和提取 (pull) 請求：github.com/aws/aws-xray-daemon

在 AWS Lambda 和 上 AWS Elastic Beanstalk，使用這些服務與 X-Ray 的整合來執行協助程式。Lambda 會在針對取樣請求叫用函數時自動執行協助程式。在 Elastic Beanstalk 上，使用`XRayEnabled`組態選項在環境中的執行個體上執行協助程式。如需詳細資訊，請參閱

若要在本機、內部部署或其他位置執行 X-Ray 協助程式 AWS 服務，請下載它、執行它，然後授予它將區段文件上傳到 X-Ray 的許可。

下載精靈

您可以從 Amazon S3、Amazon ECR 或 Docker Hub 下載協助程式，然後在本機執行，或在啟動時將其安裝在 Amazon EC2 執行個體上。

Amazon S3

X-Ray 協助程式安裝程式和可執行檔

- Linux （ 可執行檔 ） – `aws-xray-daemon-linux-3.x.zip`(sig)
- Linux (RPM 安裝程式 ） – `aws-xray-daemon-3.x.rpm`
- Linux (DEB 安裝程式 ） – `aws-xray-daemon-3.x.deb`
- Linux (ARM64，可執行檔 ） – `aws-xray-daemon-linux-arm64-3.x.zip`(sig)

- Linux (ARM64、RPM 安裝程式） – aws-xray-daemon-arm64-3.x.rpm
- Linux (ARM64、DEB 安裝程式） – aws-xray-daemon-arm64-3.x.deb
- OS X （可執行檔） – aws-xray-daemon-macos-3.x.zip(sig)
- Windows （可執行檔） – aws-xray-daemon-windows-process-3.x.zip(sig)
- Windows （服務） – aws-xray-daemon-windows-service-3.x.zip(sig)

這些連結一律指向 協助程式的最新 3.x 版本。若要下載特定版本，請執行下列動作：

- 如果您想要下載版本 之前的版本3.3.0，請將 取代3.x為版本編號。例如：2.1.0。在版本 之前3.3.0，唯一可用的架構是 arm64。例如，2.1.0 和 arm64。
- 如果您想要下載版本 之後的版本3.3.0，請將 取代3.x為版本編號，並將 arch取代為架構類型。

X-Ray 資產會複寫到每個支援區域中的儲存貯體。若要使用最接近您或您的 AWS 資源的儲存貯體，請將上述連結中的區域取代為您的區域。

```
https://s3.us-west-2.amazonaws.com/aws-xray-assets.us-west-2/xray-daemon/aws-xray-daemon-3.x.rpm
```

Amazon ECR

從 3.2.0 版開始， 協助程式可在 Amazon ECR 上找到。提取映像之前，您應該向 Amazon ECR 公有登錄檔驗證您的 Docker 用戶端。

執行下列命令以提取最新發行的 3.x 版本標籤：

```
docker pull public.ecr.aws/xray/aws-xray-daemon:3.x
```

先前或 Alpha 版本可以透過將 取代3.x為 alpha或特定版本編號來下載。不建議在生產環境中使用具有 Alpha 標籤的協助程式映像。

Docker Hub

您可以在 Docker Hub 上找到協助程式。若要下載最新發行的 3.x 版本，請執行下列命令：

```
docker pull amazon/aws-xray-daemon:3.x
```

先前的 協助程式版本可以透過將 取代3.x為所需的版本來發行。

驗證精靈存檔的簽章

GPG 簽章檔案會針對以 ZIP 存檔形式壓縮的精靈資產包含在其中。公有金鑰位於此處：aws-xray.gpg。

您可以使用公有金鑰來驗證精靈的 ZIP 存檔是否為原始狀態且未經修改。首先，透過 GnuPG 匯入公有金鑰。

匯入公有金鑰

1. 下載公開金鑰。

```
$ BUCKETURL=https://s3.us-east-2.amazonaws.com/aws-xray-assets.us-east-2
$ wget $BUCKETURL/xray-daemon/aws-xray.gpg
```

2. 將公開金鑰匯入至您的 keyring。

```
$ 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
```

使用匯入的金鑰驗證精靈 ZIP 存檔的簽章。

驗證存檔的簽章

1. 下載存檔及簽章檔案

```
$ 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. 執行 gpg --verify 以驗證簽章。

```
$ 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
```

請注意有關信任的警告。只有您或您信任的人員已進行簽署，金鑰才會被視為受信任。這不表示該簽章是無效，只是您尚未驗證該公開金鑰。

執行精靈

從命令列於本機執行精靈。使用 -o 選向來在本機模式中執行，以及 -n 來設定區域。

```
~/Downloads$ ./xray -o -n us-east-2
```

如需詳細的平台限定說明，請參閱以下主題：

- Linux（本機）– 在 Linux 上執行 X-Ray 協助程式
- Windows（本機）– 在 Windows 上執行 X-Ray 協助程式
- Elastic Beanstalk – 在 上執行 X-Ray 協助程式 AWS Elastic Beanstalk
- Amazon EC2 – 在 Amazon EC2 上執行 X-Ray 協助程式
- Amazon ECS – 在 Amazon ECS 上執行 X-Ray 協助程式

您可以使用命令列選項或組態檔來更進一步自訂精靈的行為。如需詳細資訊，請參閱 設定 AWS X-Ray 協助程式。

授予協助程式將資料傳送至 X-Ray 的許可

X-Ray 協助程式使用 AWS SDK 將追蹤資料上傳至 X-Ray，而且需要 AWS 具有許可的登入資料才能執行此操作。

在 Amazon EC2 上，協助程式會自動使用執行個體的執行個體描述檔角色。如需在本機執行協助程式所需的登入資料資訊，請參閱在本機執行您的應用程式。

若您在超過一個位置指定登入資料 (登入資料檔案、執行個體描述檔，或是環境變數)，軟體開發套件提供者鏈會判斷使用的登入資料有哪些。如需提供登入資料給 SDK 的詳細資訊，請參閱《適用於 AWS Go 的 SDK 開發人員指南》中的指定登入資料。

精靈登入資料所屬的 IAM 角色或使用者必須具備代您將資料寫入服務的許可。

- 若要在 Amazon EC2 上使用 協助程式，請建立新的執行個體描述檔角色，或將 受管政策新增至現有的執行個體描述檔角色。
- 若要在 Elastic Beanstalk 上使用 協助程式，請將 受管政策新增至 Elastic Beanstalk 預設執行個體描述檔角色。
- 若要在本機執行協助程式，請參閱在本機執行您的應用程式。

如需詳細資訊，請參閱的身分和存取管理 AWS X-Ray。

X-Ray 協助程式日誌

協助程式會輸出有關其目前組態及其傳送之區段的資訊 AWS X-Ray。

```
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)
```

根據預設，精靈會將日誌輸出到 STDOUT。若您在背景執行精靈，請使用 `--log-file` 命令列選項或組態檔來設定日誌檔案路徑。您可以設定日誌層級及停用日誌輪換。如需說明，請參閱 設定 AWS X-Ray 協助程式。

在 Elastic Beanstalk 上，平台會設定協助程式日誌的位置。如需詳細資訊，請參閱 在 上執行 X-Ray 協助程式 AWS Elastic Beanstalk。

設定 AWS X-Ray 協助程式

您可以使用命令列選項或組態檔案來自訂 X-Ray 協助程式的行為。大多數的選項皆可透過這兩種方法取得，但有些選項僅可透過組態檔取得，有些選項也僅能透過命令列使用。

若要開始使用，您需要知道的唯一選項是 `-n`或 `--region`，您可用來設定協助程式用來將追蹤資料傳送至 X-Ray 的區域。

```
~/xray-daemon$ ./xray -n us-east-2
```

如果您在本機執行協助程式，也就是說，而不是在 Amazon EC2 上，您可以新增略過檢查執行個體描述檔登入資料`-o`的選項，以便協助程式更快地準備就緒。

```
~/xray-daemon$ ./xray -o -n us-east-2
```

其餘命令列選項可讓您設定記錄日誌、在不同連接埠進行接聽、限制精靈能使用的記憶體數量，或是取得角色來將追蹤資料傳送至不同帳戶。

您可以將組態檔案傳遞給協助程式以存取進階組態選項，並執行像是限制對 X-Ray 的並行呼叫數、停用日誌輪換，以及將流量傳送至代理等動作。

章節

支援的環境變數

X-Ray 協助程式支援下列環境變數：

- AWS_REGION – 指定 X-Ray 服務端點AWS 區域的 。
- HTTPS_PROXY – 指定協助程式上傳區段的代理地址。這可以是 DNS 網域名稱，或是代理伺服器使用的 IP 地址和連接埠號碼。

使用命令列選項

在您於本機執行，或是使用使用者資料指令碼時將這些選項傳遞至精靈。

命令列選項

- -b， --bind – 接聽不同 UDP 連接埠上的區段文件。

  ```
  --bind "127.0.0.1:3000"
  ```

  預設 – 2000。
- -t， --bind-tcp – 接聽對不同 TCP 連接埠上 X-Ray 服務的呼叫。

  ```
  -bind-tcp "127.0.0.1:3000"
  ```

  預設 – 2000。

- `-c`, `--config` – 從指定的路徑載入組態檔案。

```
--config "/home/ec2-user/xray-daemon.yaml"
```

- `-f`, `--log-file` – 將日誌輸出至指定的檔案路徑。

```
--log-file "/var/log/xray-daemon.log"
```

- `-l`、`--log-level` – 日誌層級，從最詳細到最不詳細：dev、debug、 info、warn、error、prod。

```
--log-level warn
```

  預設 – prod

- `-m`, `--buffer-memory` – 變更緩衝區可以使用的記憶體量 （最低 3)。

```
--buffer-memory 50
```

  預設 – 1% 的可用記憶體。

- `-o`, `--local-mode` – 請勿檢查 EC2 執行個體中繼資料。
- `-r`, `--role-arn` – 擔任指定的 IAM 角色，將客群上傳到不同的帳戶。

```
--role-arn "arn:aws:iam::123456789012:role/xray-cross-account"
```

- `-a`, `--resource-arn` – 執行協助程式之 AWS 資源的 Amazon Resource Name (ARN)。
- `-p`, `--proxy-address` – AWS X-Ray 透過代理將區段上傳至 。必須指定代理伺服器的通訊協定。

```
--proxy-address "http://192.0.2.0:3000"
```

- `-n`, `--region` – 將區段傳送至特定區域的 X-Ray 服務。
- `-v`, `--version` – AWS X-Ray 顯示協助程式版本。
- `-h`, `--help` – 顯示說明畫面。

使用組態檔

您也可以使用 YAML 格式的檔案來設定精靈。使用 `-c` 選項來將組態檔傳遞至精靈。

```
~$ ./xray -c ~/xray-daemon.yaml
```

組態檔選項

- TotalBufferSizeMB – 緩衝區大小上限，以 MB 為單位 （ 下限 3)。選擇 0 來使用主機記憶體的 1%。
- Concurrency – AWS X-Ray 要上傳區段文件的並行呼叫數目上限。
- Region – 將客群傳送至特定區域中 AWS X-Ray 的服務。
- Socket – 設定協助程式的繫結。
  - UDPAddress – 變更協助程式接聽所在的連接埠。
  - TCPAddress – 在不同的 TCP 連接埠上接聽對 X-Ray 服務的呼叫。
- Logging – 設定記錄行為。
  - LogRotation – 設定為 false以停用日誌輪換。
  - LogLevel – 將日誌層級從最詳細變更為最少：dev、 debuginfo或 prod、warn、error、prod。預設值為 prod，相當於 info。
  - LogPath – 將日誌輸出至指定的檔案路徑。
- LocalMode – 設定為 true以略過檢查 EC2 執行個體中繼資料。
- ResourceARN – 執行協助程式之 AWS 資源的 Amazon Resource Name (ARN)。
- RoleARN – 擔任指定的 IAM 角色，將客群上傳到不同的帳戶。
- ProxyAddress – AWS X-Ray 透過代理將區段上傳至 。
- Endpoint – 變更協助程式傳送區段文件的 X-Ray 服務端點。
- NoVerifySSL – 停用 TLS 憑證驗證。
- Version – 協助程式組態檔案格式版本。檔案格式版本是必要欄位。

Example xray-daemon.yaml

此組態檔會將精靈的接聽連接埠變更為 3000、關閉執行個體中繼資料檢查，設定上傳區段所要使用的角色，並變更區域與記錄日誌選項。

```
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
```

在本機執行 X-Ray 協助程式

您可以在 Linux、MacOS、Windows AWS X-Ray 或 Docker 容器中於本機執行協助程式。當您開發和測試檢測應用程式時，請執行 協助程式，將追蹤資料轉送至 X-Ray。透過使用此處的說明，下載並解壓縮精靈。

在本機執行時，協助程式可以從 AWS SDK 登入資料檔案 (`.aws/credentials`在您的使用者目錄中） 或環境變數讀取登入資料。如需詳細資訊，請參閱授予協助程式將資料傳送至 X-Ray 的許可。

精靈會在連接埠 2000 接聽 UDP 資料。您可以透過使用組態檔和命令列選項來變更連接埠及其他選項。如需詳細資訊，請參閱設定 AWS X-Ray 協助程式。

在 Linux 上執行 X-Ray 協助程式

您可以從命令列執行精靈的可執行檔。使用 `-o` 選向來在本機模式中執行，以及 `-n` 來設定區域。

```
~/xray-daemon$ ./xray -o -n us-east-2
```

若要在背景執行精靈，請使用 &。

```
~/xray-daemon$ ./xray -o -n us-east-2 &
```

使用 `pkill` 終止在背景執行的精靈程序。

```
~$ pkill xray
```

在 Docker 容器中執行 X-Ray 協助程式

若要在 Docker 容器內於本機執行精靈，請將以下文字儲存到名為 `Dockerfile` 的檔案。在 Amazon ECR 上下載完整的範例映像。如需詳細資訊，請參閱下載協助程式。

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
```

使用 `docker build` 建置容器映像。

```
~/xray-daemon$ docker build -t xray-daemon .
```

使用 `docker run` 在容器中執行映像。

```
~/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
```

此命令使用下列選項：

- `--attach STDOUT` – 在終端機中檢視協助程式的輸出。
- `-v ~/.aws/:/root/.aws/:ro` – 授予容器對 `.aws` 目錄的唯讀存取權，讓容器讀取您的 AWS SDK 登入資料。
- AWS_REGION=*us-east-2* – 設定AWS_REGION環境變數，以告知協助程式要使用的區域。
- `--net=host` – 將容器連接至host網路。主機網路上的容器可彼此互相通訊，而無須透過連接埠發佈。
- `-p 2000:2000/udp` – 將機器上的 UDP 連接埠 2000 映射至容器上的相同連接埠。這並非位於相同網路上容器進行通訊的必要項目，但它可讓您從命令列將區段傳送至精靈，或是從沒有在 Docker 中執行的應用程式傳送。
- `--name xray-daemon` – 命名容器，xray-daemon而不是產生隨機名稱。

- -o（映像名稱之後）– 將 -o選項附加至在容器內執行協助程式的進入點。此選項會告知協助程式在本機模式下執行，以防止其嘗試讀取 Amazon EC2 執行個體中繼資料。

若要停止精靈，請使用 docker stop。若您變更 Dockerfile 並建置新的映像，您需要刪除現有的容器，才能使用相同名稱建立另一個容器。使用 docker rm 來刪除容器。

```
$ docker stop xray-daemon
$ docker rm xray-daemon
```

在 Windows 上執行 X-Ray 協助程式

您可以從命令列執行精靈的可執行檔。使用 -o 選向來在本機模式中執行，以及 -n 來設定區域。

```
> .\xray_windows.exe -o -n us-east-2
```

使用 PowerShell 指令碼來建立和執行精靈的服務。

Example PowerShell 指令碼 - 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 上執行 X-Ray 協助程式

您可以從命令列執行精靈的可執行檔。使用 -o 選向來在本機模式中執行，以及 -n 來設定區域。

```
~/xray-daemon$ ./xray_mac -o -n us-east-2
```

若要在背景執行精靈，請使用 &。

```
~/xray-daemon$ ./xray_mac -o -n us-east-2 &
```

使用 nohup 來防止精靈在終端機關閉時終止。

```
~/xray-daemon$ nohup ./xray_mac &
```

在 上執行 X-Ray 協助程式 AWS Elastic Beanstalk

若要將追蹤資料從應用程式轉送至 AWS X-Ray，您可以在 Elastic Beanstalk 環境的 Amazon EC2 執行個體上執行 X-Ray 協助程式。如需支援的平台清單，請參閱《 AWS Elastic Beanstalk 開發人員指南》中的設定 AWS X-Ray 偵錯。

> **Note**
>
> 精靈會使用您環境的執行個體描述檔以獲得許可。如需將許可新增至 Elastic Beanstalk 執行個體描述檔的說明，請參閱 授予協助程式將資料傳送至 X-Ray 的許可。

Elastic Beanstalk 平台提供組態選項，您可以將其設定為自動執行協助程式。您可以在原始程式碼的組態檔案中啟用協助程式，或在 Elastic Beanstalk 主控台中選擇選項。當您啟用組態選項時，精靈便會在執行個體上安裝並做為服務執行。

Elastic Beanstalk 平台上包含的版本可能不是最新版本。請參閱支援的平台主題以了解您平台組態可使用的精靈版本。

Elastic Beanstalk 不會在多容器 Docker (Amazon ECS) 平台上提供 X-Ray 協助程式。

使用 Elastic Beanstalk X-Ray 整合來執行 X-Ray 協助程式

使用 主控台來開啟 X-Ray 整合，或使用組態檔案在應用程式原始碼中設定。

在 Elastic Beanstalk 主控台中啟用 X-Ray 協助程式

1. 開啟 Elastic Beanstalk 主控台。
2. 導覽至您環境的管理主控台。
3. 選擇 Configuration (組態)。
4. 選擇 Software Settings (軟體設定)。
5. 針對 X-Ray daemon (X-Ray 精靈)，請選擇 Enabled (啟用)。
6. 選擇套用。

您可以在您的來源碼中包含組態檔，來在環境間攜帶您的組態。

Example .ebextensions/xray-daemon.config

```
option_settings:
  aws:elasticbeanstalk:xray:
    XRayEnabled: true
```

Elastic Beanstalk 會將組態檔案傳遞至協助程式，並將日誌輸出至標準位置。

Windows Server 平台

- 組態檔案 – `C:\Program Files\Amazon\XRay\cfg.yaml`
- 日誌 – `c:\Program Files\Amazon\XRay\logs\xray-service.log`

Linux 平台

- 組態檔案 – `/etc/amazon/xray/cfg.yaml`
- 日誌 – `/var/log/xray/xray.log`

Elastic Beanstalk 提供從 AWS Management Console 或 命令列提取執行個體日誌的工具。您可以透過使用組態檔案新增任務，指示 Elastic Beanstalk 包含 X-Ray 協助程式日誌。

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
```

如需詳細資訊，請參閱《 開發人員指南》中的從 Elastic Beanstalk 環境的 Amazon EC2 執行個體檢視日誌。 AWS Elastic Beanstalk

手動下載和執行 X-Ray 協助程式（進階）

如果您的平台組態無法使用 X-Ray 協助程式，您可以從 Amazon S3 下載它，並使用組態檔案執行它。

使用 Elastic Beanstalk 組態檔案下載並執行協助程式。

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
```

這些範例也會將協助程式的日誌檔案新增至 Elastic Beanstalk 結尾日誌任務，以便在您使用主控台或 Elastic Beanstalk 命令列界面 (EB CLI) 請求日誌時包含該檔案。

在 Amazon EC2 上執行 X-Ray 協助程式

您可以在 Amazon EC2 的下列作業系統上執行 X-Ray 協助程式：

- Amazon Linux
- Ubuntu
- Windows Server (2012 R2 和更新版本)

使用執行個體描述檔授予協助程式上傳追蹤資料的許可到 X-Ray。如需詳細資訊，請參閱[授予協助程式將資料傳送至 X-Ray 的許可]。

利用使用者資料指令碼，在您啟動執行個體時自動執行協助程式。

Example 使用者資料指令碼 - 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 使用者資料指令碼 - 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 上執行 X-Ray 協助程式

在 Amazon ECS 中，建立執行 X-Ray 協助程式的 Docker 映像，將其上傳至 Docker 映像儲存庫，然後將其部署到您的 Amazon ECS 叢集。您可以使用您任務定義檔案中的連接埠映射和網路模式設定，來允許應用程式與精靈容器通訊。

使用官方 Docker 映像檔

X-Ray 在 Amazon ECR 上提供 Docker 容器映像，您可以隨應用程式一起部署。如需詳細資訊，請參閱下載協助程式。

Example 任務定義

```
{
  "name": "xray-daemon",
  "image": "amazon/aws-xray-daemon",
  "cpu": 32,
  "memoryReservation": 256,
  "portMappings" : [
      {
          "hostPort": 0,
          "containerPort": 2000,
          "protocol": "udp"
      }
   ]
}
```

建立和建置 Docker 影像

用於自訂組態時，您可能需要定義自己的 Docker 影像。

將受管政策新增至您的任務角色，以授予協助程式將追蹤資料上傳至 X-Ray 的許可。如需詳細資訊，請參閱授予協助程式將資料傳送至 X-Ray 的許可。

使用以下其中一個 Dockerfiles 建立執行精靈的映像。

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**
>
> 指定要接聽多重容器環境迴路的繫結地址時，必須提供標記 `-t` 和 `-b`。

Example Dockerfile – Ubuntu

若是 Debian 的衍生產品，您還需要安裝憑證授權機構 (CA) 憑證，以避免下載安裝程式時發生問題。

```
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
```

在您的任務定義中，組態取決於您使用的聯網模式。預設值是橋接聯網，可在您的預設 VPC 中使用。在橋接網路中，設定 AWS_XRAY_DAEMON_ADDRESS 環境變數，以告知 X-Ray SDK 要參考和設定主機連接埠的容器連接埠。舉例來說，您可以發佈 UDP 連接埠 2000，然後建立您應用程式容器與精靈容器的連結。

Example 任務定義

```
{
  "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"
      ]
    }
```

若您在 VPC 私有子網路中執行您的叢集，您可以使用 awsvpc 網路模式來將彈性網路界面 (ENI) 連接到您的容器。這可讓您避免使用連結。忽略連接埠映射、連結，以及 AWS_XRAY_DAEMON_ADDRESS 環境變數中的主機連接埠。

Example VPC 任務定義

```
{
    "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 主控台中設定命令列選項

命令列選項會覆寫影像組態檔中的任何衝突值。命令列選項通常用於本機測試，但也可以在設定環境變數或控制啟動程序時使用。

透過新增命令列選項，您將更新傳遞給容器的 Docker CMD。如需詳細資訊，請參閱 Docker 執行參考。

若要設定命令列選項

1. 開啟 Amazon ECS 傳統主控台，網址為 https://console.aws.amazon.com/ecs/。
2. 從導覽列中選擇包含您任務定義的區域。
3. 在導覽窗格中，選擇 Task Definitions (任務定義)。
4. 在 Task Definitions (任務定義) 頁面中，選取要修訂之任務定義左側的方塊，並選擇 Create new revision (建立新修訂版)。
5. 在 Create new revision of Task Definition (建立任務定義的新修訂版) 頁面上，選取容器。
6. 在 ENVIRONMENT (環境) 區段中，將逗號分隔的命令列選項清單新增至 Command (命令) 欄位。
7. 選擇更新。
8. 驗證資訊，然後選擇 Create (建立)。

下列範例顯示如何為 RoleARN 選項撰寫逗號分隔的命令列。RoleARN 選項會擔任指定的 IAM 角色，將區段上傳至不同的帳戶。

Example

```
--role-arn, arn:aws:iam::123456789012:role/xray-cross-account
```

若要進一步了解 X-Ray 中可用的命令列選項，請參閱設定 AWS X-Ray 協助程式。

AWS X-Ray 與其他 整合 AWS 服務

許多 AWS 服務 提供不同層級的 X-Ray 整合，包括取樣和將標頭新增至傳入請求、執行 X-Ray 協助程式，以及自動將追蹤資料傳送至 X-Ray。與 X-Ray 整合可以包括下列項目：

- 主動檢測 – 取樣和檢測傳入請求
- 被動檢測 – 檢測由其他 服務取樣的請求
- 請求追蹤 – 將追蹤標頭新增至所有傳入請求，並將其傳播至下游
- 工具 – 執行 X-Ray 協助程式以從 X-Ray SDK 接收區段

> **Note**
>
> X-Ray SDKs包含可與其他 整合的外掛程式 AWS 服務。例如，您可以使用適用於 Java Elastic Beanstalk 的 X-Ray 開發套件外掛程式來新增執行應用程式之 Elastic Beanstalk 環境的相關資訊，包括環境名稱和 ID。

以下是與 X-Ray AWS 服務 整合的一些範例：

- AWS Distro for OpenTelemetry (ADOT) – 透過 ADOT，工程師可以檢測一次其應用程式，並將相關指標和追蹤傳送至多個 AWS 監控解決方案，包括 Amazon CloudWatch AWS X-Ray、Amazon OpenSearch Service 和 Amazon Managed Service for Prometheus。
- AWS Lambda – 主動和被動檢測所有執行時間的傳入請求。 會將兩個節點 AWS Lambda 新增至追蹤映射，一個用於 AWS Lambda 服務，另一個用於 函數。當您啟用檢測時， AWS Lambda 也會在 Java 和 Node.js 執行時間上執行 X-Ray 協助程式，以便與 X-Ray SDK 搭配使用。
- Amazon API Gateway – 主動和被動檢測。API Gateway 使用取樣規則來判斷要記錄的請求，並將閘道階段的節點新增至您的服務映射。
- AWS Elastic Beanstalk – 工具。Elastic Beanstalk 包含下列平台上的 X-Ray 協助程式：
  - Java SE – 2.3.0 及更新版本組態
  - Tomcat – 2.4.0 和更新版本的組態
  - Node.js – 3.2.0 和更新版本的組態
  - Windows Server – 2016 年 12 月 9 日之後發行的 Windows Server Core 以外的所有組態

  您可以使用 Elastic Beanstalk 主控台，指示 Elastic Beanstalk 在這些平台上執行協助程式，或使用 `aws:elasticbeanstalk:xray` 命名空間中的 `XRayEnabled`選項。

- Elastic Load Balancing – 在 Application Load Balancer 上請求追蹤。Application Load Balancer 會將追蹤 ID 新增至請求標頭，然後再將其傳送至目標群組。
- Amazon EventBridge – 被動檢測。如果使用 X-Ray SDK 檢測將事件發佈至 EventBridge 的服務，事件目標會收到追蹤標頭，並且可以繼續傳播原始追蹤 ID。
- Amazon Simple Notification Service – 被動式檢測。如果 Amazon SNS 發佈者使用 X-Ray 開發套件追蹤其用戶端，訂閱者可以擷取追蹤標頭，並繼續從發佈者傳播具有相同追蹤 ID 的原始追蹤。
- Amazon Simple Queue Service – 被動檢測。如果服務使用 X-Ray SDK 追蹤請求，Amazon SQS 可以傳送追蹤標頭，並繼續使用一致的追蹤 ID 將原始追蹤從寄件者傳播給消費者。

從下列主題中選擇，以探索整組整合的 AWS 服務。

主題

- AWS Distro for OpenTelemetry 和 AWS X-Ray
- 的 Amazon API Gateway 主動追蹤支援 AWS X-Ray
- Amazon EC2 和 AWS App Mesh
- AWS App Runner 和 X-Ray
- AWS AppSync 而且 AWS X-Ray
- 使用 記錄 X-Ray API 呼叫 AWS CloudTrail
- CloudWatch 與 X-Ray 整合
- 使用 追蹤 X-Ray 加密組態變更 AWS Config
- Amazon Elastic Compute Cloud 和 AWS X-Ray
- AWS Elastic Beanstalk 而且 AWS X-Ray
- Elastic Load Balancing 和 AWS X-Ray
- Amazon EventBridge 和 AWS X-Ray
- AWS Lambda 而且 AWS X-Ray
- Amazon SNS 和 AWS X-Ray
- AWS Step Functions 而且 AWS X-Ray
- Amazon SQS 和 AWS X-Ray
- Amazon S3 和 AWS X-Ray

AWS Distro for OpenTelemetry 和 AWS X-Ray

使用 AWS Distro for OpenTelemetry (ADOT) 收集指標和追蹤，並將其傳送至 AWS X-Ray 和其他監控解決方案，例如 Amazon CloudWatch、Amazon OpenSearch Service 和 Amazon Managed Service for Prometheus。

AWS Distro for OpenTelemetry

AWS Distro for OpenTelemetry (ADOT) 是基於雲端原生運算基金會 (CNCF) OpenTelemetry 專案的 AWS 分佈。OpenTelemetry 提供一組開放原始碼 APIs、程式庫和代理程式，以收集分散式追蹤和指標。此工具組是上游 OpenTelemetry 元件的分佈，包括 SDKs、自動檢測代理程式和 測試、最佳化、保護和支援的收集器 AWS。

透過 ADOT，工程師可以一次檢測其應用程式，並將相關指標和追蹤傳送至多個 AWS 監控解決方案 AWS X-Ray，包括 Amazon CloudWatch、Amazon OpenSearch Service 和 Amazon Managed Service for Prometheus。

ADOT 與越來越多的 整合 AWS 服務 ，可簡化傳送追蹤和指標以監控解決方案，例如 X-Ray。與 ADOT 整合的一些服務範例包括：

- AWS Lambda – 適用於 ADOT 的 AWS 受管 Lambda 層透過自動檢測 Lambda 函數、將 OpenTelemetry 與 AWS Lambda 和 X-Ray out-of-the-box組態封裝在易於設定的層中，提供plug-and-play使用者體驗。使用者可以為其 Lambda 函數啟用和停用 OpenTelemetry，而無需變更程式碼。如需詳細資訊，請參閱 AWS Distro for OpenTelemetry Lambda
- Amazon Elastic Container Service (ECS) – 使用 AWS Distro for OpenTelemetry Collector 從 Amazon ECS 應用程式收集指標和追蹤，以傳送至 X-Ray 和其他監控解決方案。如需詳細資訊，請參閱《Amazon ECS 開發人員指南》中的收集應用程式追蹤資料。
- AWS App Runner – App Runner 支援使用 AWS Distro for OpenTelemetry (ADOT) 將追蹤傳送至 X-Ray。使用 ADOT SDKs來收集容器化應用程式的追蹤資料，並使用 X-Ray 來分析並深入了解經檢測的應用程式。如需詳細資訊，請參閱 AWS App Runner 和 X-Ray。

如需 AWS Distro for OpenTelemetry 的詳細資訊，包括與其他 的整合 AWS 服務，請參閱 AWS Distro for OpenTelemetry 文件。

如需使用 AWS Distro for OpenTelemetry 和 X-Ray 檢測應用程式的詳細資訊，請參閱使用 AWS Distro for OpenTelemetry 檢測應用程式。

的 Amazon API Gateway 主動追蹤支援 AWS X-Ray

您可以使用 X-Ray 在使用者請求通過 Amazon API Gateway APIs 到基礎服務時追蹤和分析使用者請求。API Gateway 支援所有 API Gateway 端點類型的 X-Ray 追蹤：區域、邊緣最佳化和私有。您可以在可使用 X-Ray 的所有 AWS 區域 中使用 X-Ray 搭配 Amazon API Gateway。如需詳細資訊，請參閱《Amazon API Gateway API Gateway 開發人員指南》中的使用 追蹤 API Gateway API 執行 AWS X-Ray。

> **Note**
>
> X-Ray 僅支援透過 APIs Gateway 追蹤 REST API。

Amazon API Gateway 提供 的主動追蹤支援 AWS X-Ray。在您的 API 階段上啟用主動追蹤，以取樣傳入的請求，並將追蹤傳送至 X-Ray。

啟用 API 階段的主動追蹤

1. 在以下網址開啟 API Gateway 主控台：https://console.aws.amazon.com/apigateway/。
2. 選擇一個 API。
3. 選擇一個階段。
4. 在日誌/追蹤索引標籤上，選擇啟用 X-Ray 追蹤，然後選擇儲存變更。
5. 在左側導覽窗格中，選擇 Resources (資源)。
6. 若要使用新設定重新部署 API，請選擇動作下拉式清單，然後選擇部署 API。

API Gateway 會使用您在 X-Ray 主控台中定義的抽樣規則來決定要記錄哪些請求。您可以建立僅適用於 APIs規則，或僅適用於包含特定標頭的請求的規則。API Gateway 會在區段的屬性中記錄標頭，以及階段和請求的詳細資訊。如需詳細資訊，請參閱設定 取樣規則。

> **Note**
>
> 使用 API Gateway HTTP 整合追蹤 REST APIs 時，每個區段的服務名稱會設定為從 API Gateway 到 HTTP 整合端點的請求 URL 路徑，導致每個唯一 URL 路徑的 X-Ray 追蹤映射上有一個服務節點。大量的 URL 路徑可能會導致追蹤映射超過 10，000 個節點的限制，進而導致錯誤。

> 若要將 API Gateway 建立的服務節點數量降至最低，請考慮在 URL 查詢字串內或透過 POST 在請求內文中傳遞參數。這兩種方法都會確保參數不屬於 URL 路徑，這可能會導致不同的 URL 路徑和服務節點較少。

對於所有傳入請求，API Gateway 會將追蹤標頭新增至尚未有的傳入 HTTP 請求。

```
X-Amzn-Trace-Id: Root=1-5759e988-bd862e3fe1be46a994272793
```

X-Ray 追蹤 ID 格式

X-Ray `trace_id`由以連字號分隔的三個數字組成。例如：`1-58406520-a006649127e371903a2de979`。其中包含：

- 版本編號，即 1。
- 使用 8 個十六進位數字的 Unix epoch 時間原始請求的時間。

  例如，太平洋標準時間 2016 年 12 月 1 日上午 10：00，以秒`1480615200`為單位或以十六進位數字`58406520`為單位。
- 追蹤的全域唯一 96 位元識別符，以 24 個十六進位數字表示。

即使停用主動追蹤，如果請求是來自已抽樣請求並已開始追蹤的服務，階段仍會記錄區段。例如，經檢測的 Web 應用程式可以使用 HTTP 用戶端呼叫 API Gateway API。當您使用 X-Ray SDK 檢測 HTTP 用戶端時，它會將追蹤標頭新增至包含抽樣決策的傳出請求。API Gateway 會讀取追蹤標頭，並為取樣的請求建立區段。

如果您使用 API Gateway 為您的 API 產生 Java SDK，您可以透過使用用戶端建置器新增請求處理常式來檢測 SDK 用戶端，方法與手動檢測 AWS SDK 用戶端的方式相同。如需說明，請參閱 使用適用於 Java 的 X-Ray AWS 開發套件追蹤 SDK 呼叫。

Amazon EC2 和 AWS App Mesh

AWS X-Ray 與 整合AWS App Mesh，以管理微服務的 Envoy 代理。App Mesh 提供 Envoy 的版本，您可以設定此版本將追蹤資料傳送至在相同任務或 Pod 的容器中執行的 X-Ray 協助程式。X-Ray 支援使用下列 App Mesh 相容服務進行追蹤：

- Amazon Elastic Container Service (Amazon ECS)

- Amazon Elastic Kubernetes Service (Amazon EKS)
- Amazon Elastic Compute Cloud (Amazon EC2)

使用以下指示，了解如何透過 App Mesh 來啟用 X-Ray 追蹤。

若要設定 Envoy 代理將資料傳送至 X-Ray，請在其容器定義中設定`ENABLE_ENVOY_XRAY_TRACING`環境變數。

> **Note**
>
> Envoy 的 App Mesh 版本目前不會根據設定的取樣規則傳送追蹤。相反地，它使用 Envoy 1.16.3 版或更新版本的固定取樣率為 5%，或 Envoy 1.16.3 版之前的 50% 取樣率。

Example Amazon ECS 的 Envoy 容器定義

```
{
      "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**
>
> 若要進一步了解可用的 Envoy 區域地址，請參閱 AWS App Mesh 《使用者指南》中的 Envoy 映像。

如需在容器中執行 X-Ray 協助程式的詳細資訊，請參閱 在 Amazon ECS 上執行 X-Ray 協助程式。對於包含服務網格、微服務、Envoy 代理和 X-Ray 協助程式的範例應用程式，請在 App Mesh 範例 GitHub 儲存庫中部署`colorapp`範例。

進一步了解

- 開始使用 AWS App Mesh
- AWS App Mesh 和 Amazon ECS 入門

AWS App Runner 和 X-Ray

AWS App Runner 是 AWS 服務 ，提供快速、簡單且符合成本效益的方式，可將原始碼或容器映像直接部署到 中可擴展且安全的 Web 應用程式 AWS 雲端。您不需要學習新技術、決定要使用的運算服務，或知道如何佈建和設定 AWS 資源。如需詳細資訊，請參閱什麼是 AWS App Runner。

AWS App Runner 透過與 AWS Distro for OpenTelemetry (ADOT) 整合，將追蹤傳送至 X-Ray。使用 ADOT SDKs來收集容器化應用程式的追蹤資料，並使用 X-Ray 來分析並深入了解經檢測的應用程式。如需詳細資訊，請參閱使用 X-Ray 追蹤 App Runner 應用程式。

AWS AppSync 而且 AWS X-Ray

您可以啟用和追蹤 AWS AppSync 的請求。如需詳細資訊，請參閱使用 AWS X-Ray 追蹤以取得指示。

為 an AWS AppSync API 啟用 X-Ray 追蹤時，會在您的帳戶中自動建立 AWS Identity and Access Management 服務連結角色，並具有適當的許可。這可讓 AWS AppSync 以安全的方式將追蹤傳送至 X-Ray。

使用 記錄 X-Ray API 呼叫 AWS CloudTrail

AWS X-Ray 已與 整合AWS CloudTrail，此服務提供使用者、角色或 所採取動作的記錄 AWS 服務。CloudTrail 會將 X-Ray 的所有 API 呼叫擷取為事件。擷取的呼叫包括從 X-Ray 主控台的呼叫，以及對 X-Ray API 操作的程式碼呼叫。您可以使用 CloudTrail 所收集的資訊，判斷對 X-Ray 提出的請求、提出請求的 IP 地址、提出請求的時間，以及其他詳細資訊。

每一筆事件或日誌專案都會包含產生請求者的資訊。身分資訊可協助您判斷下列事項：

- 該請求是使用根使用者還是使用者憑證提出。

- 請求是否代表 IAM Identity Center 使用者提出。
- 提出該請求時，是否使用了特定角色或聯合身分使用者的暫時安全憑證。
- 該請求是否由另一項 AWS 服務服務提出。

當您建立帳戶 AWS 帳戶 時CloudTrail 會在 中處於作用中狀態，而且您會自動存取 CloudTrail 事件歷史記錄。CloudTrail 事件歷史記錄為 AWS 區域中過去 90 天記錄的管理事件，提供可檢視、可搜尋、可下載且不可變的記錄。如需詳細資訊，請參閱「AWS CloudTrail 使用者指南」中的使用 CloudTrail 事件歷史記錄。檢視事件歷史記錄不會產生 CloudTrail 費用。

如需 AWS 帳戶 過去 90 天內持續記錄的事件，請建立線索或 CloudTrail Lake 事件資料存放區。

CloudTrail 追蹤

線索能讓 CloudTrail 將日誌檔案交付至 Amazon S3 儲存貯體。使用 建立的所有線索 AWS Management Console 都是多區域。您可以使用 AWS CLI建立單一或多區域追蹤。建議您建立多區域線索，因為您擷取 AWS 區域 帳戶中所有 的活動。如果您建立單一區域追蹤，您只能檢視追蹤 AWS 區域中記錄的事件。如需追蹤的詳細資訊，請參閱《AWS CloudTrail 使用者指南》中的為您的 AWS 帳戶建立追蹤和為組織建立追蹤。

您可以透過建立追蹤，免費將持續管理事件的一個複本從 CloudTrail 傳遞至您的 Amazon S3 儲存貯體，但這樣做會產生 Amazon S3 儲存費用。如需 CloudTrail 定價的詳細資訊，請參閱 AWS CloudTrail 定價。如需 Amazon S3 定價的相關資訊，請參閱 Amazon S3 定價。

CloudTrail Lake 事件資料存放區

CloudTrail Lake 讓您能夠對事件執行 SQL 型查詢。CloudTrail Lake 會將分列式 JSON 格式的現有事件轉換為 Apache ORC 格式。ORC 是一種單欄式儲存格式，針對快速擷取資料進行了最佳化。系統會將事件彙總到事件資料存放區中，事件資料存放區是事件的不可變集合，其依據為您透過套用進階事件選取器選取的條件。套用於事件資料存放區的選取器控制哪些事件持續存在並可供您查詢。如需 CloudTrail Lake 的詳細資訊，請參閱AWS CloudTrail 《 使用者指南》中的使用 AWS CloudTrail Lake。

CloudTrail Lake 事件資料存放區和查詢會產生費用。建立事件資料存放區時，您可以選擇要用於事件資料存放區的定價選項。此定價選項將決定擷取和儲存事件的成本，以及事件資料存放區的預設和最長保留期。如需 CloudTrail 定價的詳細資訊，請參閱 AWS CloudTrail 定價。

主題

- CloudTrail 中的 X-Ray 管理事件

- CloudTrail 中的 X-Ray 資料事件
- X-Ray 事件範例

CloudTrail 中的 X-Ray 管理事件

AWS X-Ray 與 整合， AWS CloudTrail 以記錄使用者、角色或 X-Ray AWS 服務 中的 所做的 API 動作。您可以使用 CloudTrail 即時監控 X-Ray API 請求，並將日誌存放在 Amazon S3、Amazon CloudWatch Logs 和 Amazon CloudWatch Events 中。X-Ray 支援將下列動作記錄為 CloudTrail 日誌檔案中的事件：

支援的 API 動作

- PutEncryptionConfig
- GetEncryptionConfig
- CreateGroup
- UpdateGroup
- DeleteGroup
- GetGroup
- GetGroups
- GetInsight
- GetInsightEvents
- GetInsightImpactGraph
- GetInsightSummaries
- GetSamplingStatisticSummaries

CloudTrail 中的 X-Ray 資料事件

資料事件提供有關在資源上執行或在資源中執行的資源操作的資訊（例如， PutTraceSegments會將區段文件上傳至 X-Ray)。

這些也稱為資料平面操作。資料事件通常是大量資料的活動。根據預設，CloudTrail 不會記錄資料事件。CloudTrail 事件歷史記錄不會記錄資料事件。

資料事件需支付額外的費用。如需 CloudTrail 定價的詳細資訊，請參閱 AWS CloudTrail 定價。

您可以使用 CloudTrail 主控台或 CloudTrail API 操作 AWS CLI來記錄 X-Ray 資源類型的資料事件。如需如何記錄資料事件的詳細資訊，請參閱《AWS CloudTrail 使用者指南》中的使用 AWS Management Console記錄資料事件和使用 AWS Command Line Interface記錄資料事件。

下表列出您可以記錄資料事件的 X-Ray 資源類型。資料事件類型 (主控台) 資料行會顯示從 CloudTrail 主控台上的資料事件類型清單中選擇的值。resources.type 值欄會顯示值，您會在使用 AWS CLI 或 CloudTrail APIs 設定進階事件選取器時指定該`resources.type`值。記錄到 CloudTrail 的資料 API 資料行會針對資源類型顯示記錄到 CloudTrail 的 API 呼叫。

資料事件類型 (主控台)	resources.type 值	記錄到 CloudTrail 的資料 API
X-Ray 追蹤	`AWS::XRay::Trace`	• PutTraceSegments • GetTraceSummaries • GetTraceGraph • GetServiceGraph • BatchGetTraces • GetTimeSeriesServiceStatistics • PutTelemetryRecords • GetSamplingTargets

您可以設定進階事件選取器來篩選 `eventName`和 `readOnly` 欄位，以僅記錄對您重要的事件。不過，您無法透過新增`resources.ARN`欄位選擇器來選取事件，因為 X-Ray 追蹤沒有 ARNs。如需這些欄位的詳細資訊，請參閱AWS CloudTrail API 參考中的AdvancedFieldSelector。以下是如何執行 `put-event-selectors` AWS CLI 命令以記錄 CloudTrail 追蹤資料事件的範例。您必須在 中執行 命令，或指定建立追蹤的區域；否則，操作會傳回`InvalidHomeRegionException`例外狀況。

```
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 事件範例

管理事件範例，**GetEncryptionConfig**

以下是 CloudTrail 中的 X-Ray GetEncryptionConfig日誌項目範例。

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"
}
```

資料事件範例，**PutTraceSegments**

以下是 CloudTrail 中的 X-Ray PutTraceSegments資料事件日誌項目範例。

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 與 X-Ray 整合

AWS X-Ray 與 CloudWatch Application Signals、CloudWatch RUM 和 CloudWatch Synthetics 整合，讓您更輕鬆地監控應用程式的運作狀態。啟用 Application Signals 的應用程式，以監控和疑難排解服務、用戶端頁面、Synthetics Canary 和服務相依性的運作狀態。

透過關聯 CloudWatch 指標、日誌和 X-Ray 追蹤，X-Ray 追蹤映射可提供服務的end-to-end檢視，協助您快速找出效能瓶頸並識別受影響的使用者。

使用 CloudWatch RUM，您可以執行實際使用者監控，從實際使用者工作階段近乎即時地收集和檢視 Web 應用程式效能的用戶端資料。使用 AWS X-Ray 和 CloudWatch RUM，您可以從應用程式的最終

使用者開始，透過下游 AWS 受管服務來分析和偵錯請求路徑。這樣做有助於找出影響最終使用者的延遲趨勢和錯誤。

主題

CloudWatch RUM 和 AWS X-Ray

使用 Amazon CloudWatch RUM，您可以執行實際使用者監控，從實際使用者工作階段近乎即時地收集和檢視 Web 應用程式效能的用戶端資料。使用 AWS X-Ray 和 CloudWatch RUM，您可以透過下游 AWS 受管服務，分析和偵錯從應用程式最終使用者開始的請求路徑。這樣做有助於找出影響最終使用者的延遲趨勢和錯誤。

開啟使用者工作階段的 X-Ray 追蹤後，CloudWatch RUM 會將 X-Ray 追蹤標頭新增至允許的 HTTP 請求，並針對允許的 HTTP 請求記錄 X-Ray 區段。然後，您可以在 X-Ray 和 CloudWatch 主控台中查看來自這些使用者工作階段的追蹤和客群，包括 X-Ray 追蹤映射。

> **Note**
>
> CloudWatch RUM 不會與 X-Ray 取樣規則整合。相反地，當您設定應用程式使用 CloudWatch RUM 時，請選擇取樣百分比。從 CloudWatch RUM 傳送的追蹤可能會產生額外費用。如需詳細資訊，請參閱 AWS X-Ray 定價。

根據預設，從 CloudWatch RUM 傳送的用戶端追蹤不會連接到伺服器端追蹤。若要將用戶端追蹤與伺服器端追蹤連線，請設定 CloudWatch RUM Web 用戶端，將 X-Ray 追蹤標頭新增至這些 HTTP 請求。

> **Warning**
>
> 設定 CloudWatch RUM Web 用戶端將 X-Ray 追蹤標頭新增至 HTTP 請求，可能會導致跨來源資源共用 (CORS) 失敗。若要避免這種情況，請將 `X-Amzn-Trace-Id` HTTP 標頭新增至下游服務的 CORS 組態上允許的標頭清單。如果您使用 API Gateway 做為下游，請參閱啟用 REST API 資源的 CORS。強烈建議您在生產環境中新增用戶端 X-Ray 追蹤標頭之前，先測試您的應用程式。如需詳細資訊，請參閱 CloudWatch RUM Web 用戶端文件。

如需 CloudWatch 中實際使用者監控的詳細資訊，請參閱使用 CloudWatch RUM。若要設定您的應用程式以使用 CloudWatch RUM，包括使用 X-Ray 追蹤使用者工作階段，請參閱設定應用程式以使用 CloudWatch RUM。

使用 X-Ray 偵錯 CloudWatch 合成 Canary

CloudWatch Synthetics 是一項全受管服務，可讓您使用指令碼 Canary 監控端點和 APIs，該 Canary 每天執行 24 小時，每分鐘一次。

您可以自訂 Canary 指令碼，以檢查下列項目中的變更：

- 可用性
- Latency (延遲)
- 交易
- 中斷或失效的連結
- 逐步完成任務
- 頁面載入錯誤
- UI 資產的載入延遲
- 複雜的精靈流程
- 應用程式中的簽出流程

Canary 會沿著相同的路線，執行與客戶相同的動作和行為，持續驗證客戶的體驗。

若要深入了解如何設定 Synthetics 測試，請參閱使用 Synthetics 建立及管理 Canary。

下列範例會示範您 Synthetics Canary 引起的偵錯問題常見使用案例。每個範例都會示範使用追蹤映射或 X-Ray Analytics 主控台進行偵錯的關鍵策略。

如需如何讀取追蹤映射並與之互動的詳細資訊，請參閱檢視服務映射。

如需如何讀取 X-Ray Analytics 主控台並與之互動的詳細資訊，請參閱與 AWS X-Ray Analytics 主控台互動。

主題

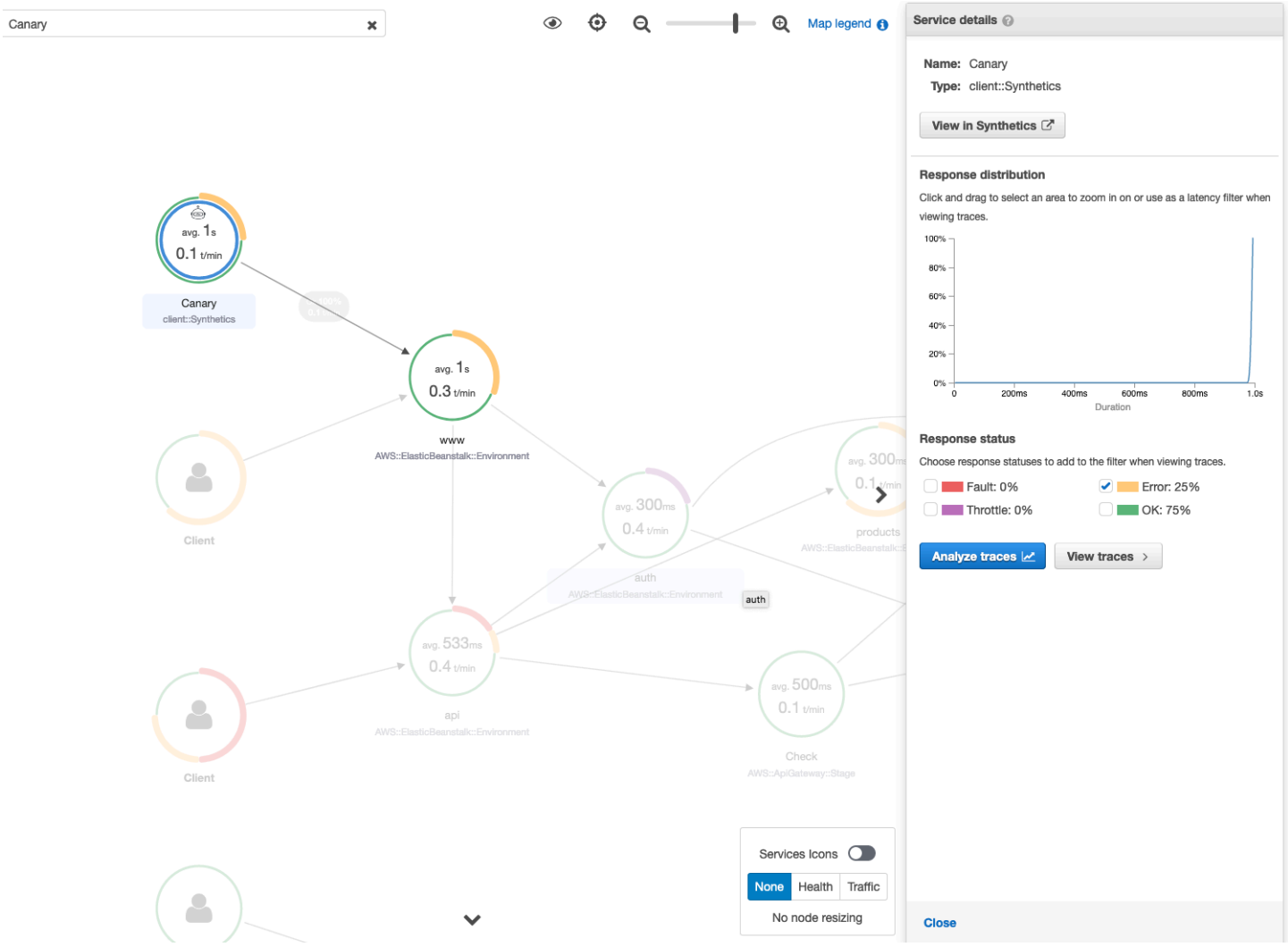
- 在追蹤映射中檢視錯誤報告增加的 Canary
- 使用個別追蹤的追蹤詳細資訊映射，詳細檢視每個請求
- 判斷上游和下游服務中持續性失敗的根本原因

- 識別效能瓶頸和趨勢
- 比較變更前後的延遲及錯誤率或容錯率
- 決定所有 API 和 URL 的必要 Canary 涵蓋範圍
- 使用群組專注於 Synthetics 測試

在追蹤映射中檢視錯誤報告增加的 Canary

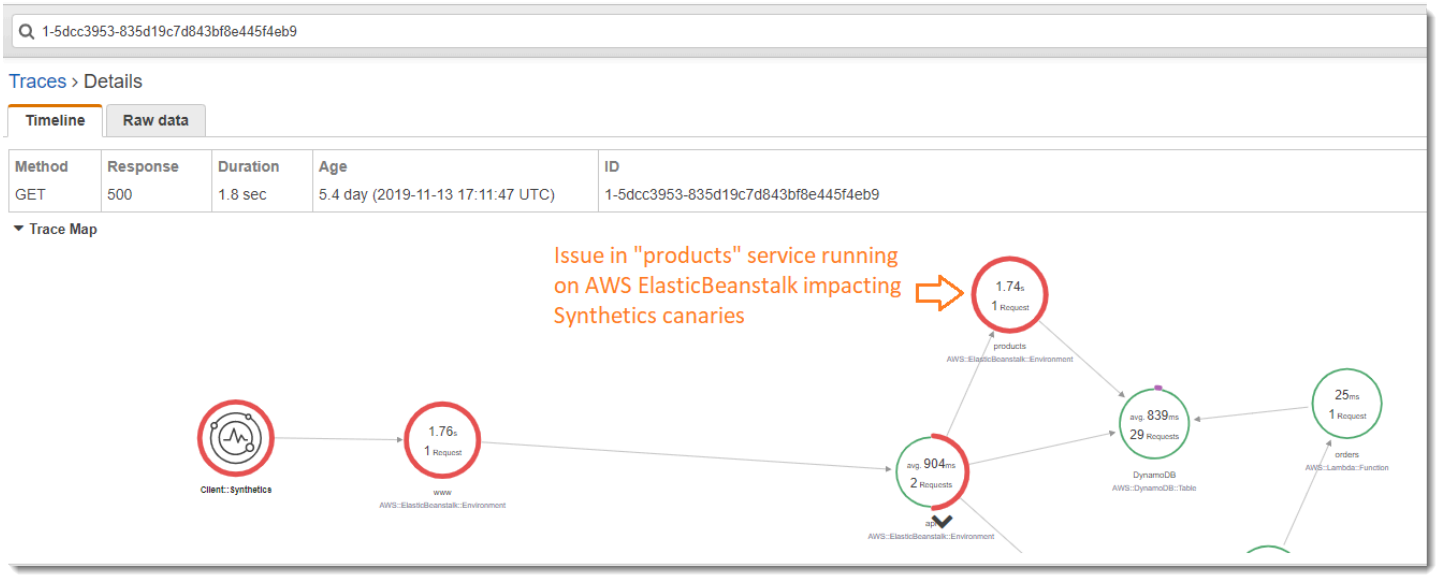
若要查看 X-Ray 追蹤映射中的哪些 Canary 的錯誤、故障、限流率或回應時間變慢，您可以使用 `Client::Synthetic` 篩選條件反白顯示 Synthetics Canary 用戶端節點。按一下節點會顯示整個請求的回應時間分佈。按一下兩個節點之間的邊緣，會顯示已傳送該連線之請求的詳細資訊。您也可以在追蹤映射中檢視相關下游服務的「遠端」推斷節點。

當您按一下 Synthetics 節點時，側邊面板上有一個檢視在 Synthetics 中按鈕，會將您重新導向至 Synthetics 主控台，您可以在其中檢查 Canary 詳細資訊。



使用個別追蹤的追蹤詳細資訊映射，詳細檢視每個請求

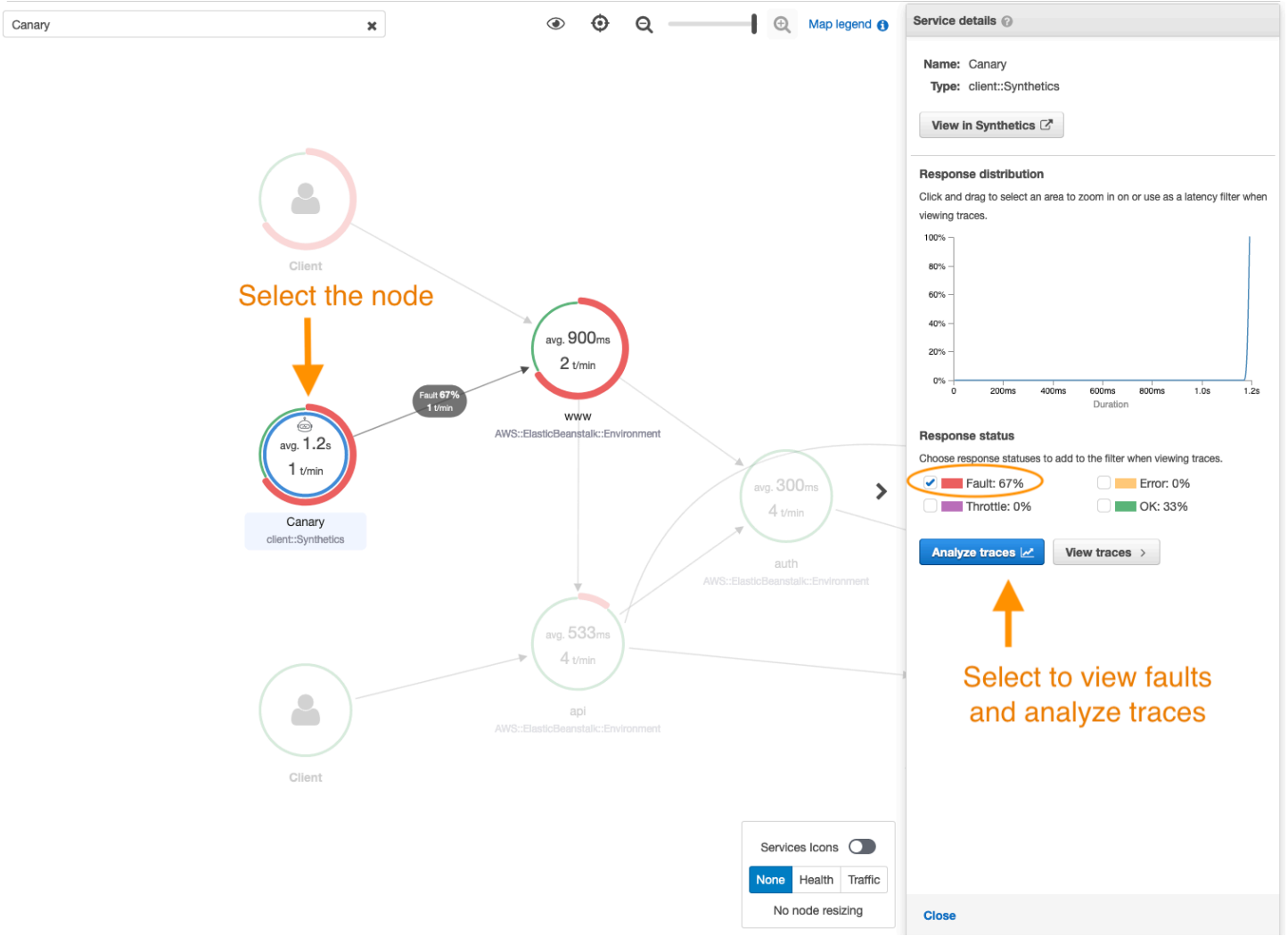
若要判斷哪些服務導致最多延遲或導致錯誤，請在追蹤映射中選取追蹤，以叫用追蹤詳細資訊映射。個別追蹤詳細資訊映射會顯示單一請求的end-to-end路徑。使用此項目可了解呼叫的服務，並視覺化上游和下游服務。

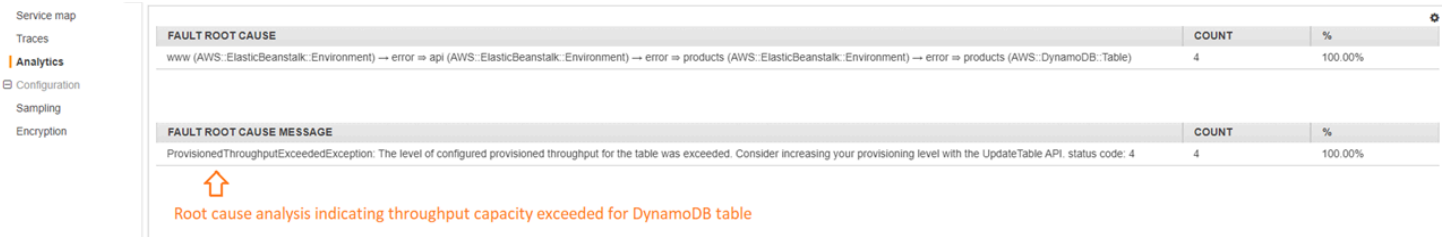


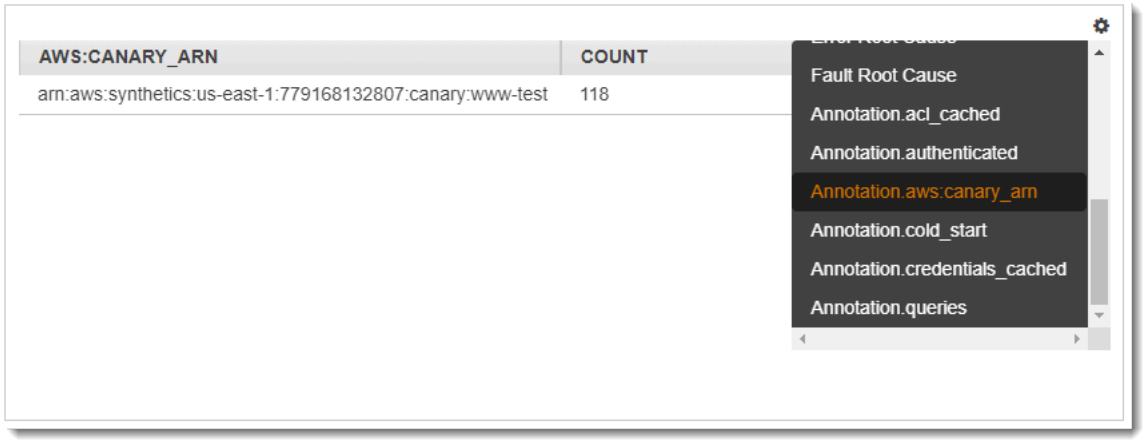
判斷上游和下游服務中持續性失敗的根本原因

收到 Synthetics Canary 中故障的 CloudWatch 警示後，請使用 X-Ray 中追蹤資料的統計建模，在 X-Ray Analytics 主控台中判斷問題的可能根本原因。在 Analytics 主控台中，回應時間根本原因資料表會顯示記錄的實體路徑。X-Ray 會判斷追蹤中哪個路徑最有可能導致回應時間。格式指出實體遇到的階層，以回應時間根本原因結束。

下列範例顯示，在 API Gateway 上執行之 API "XXX" 的 Synthetics 測試由於 Amazon DynamoDB 資料表的輸送量容量例外狀況而失敗。

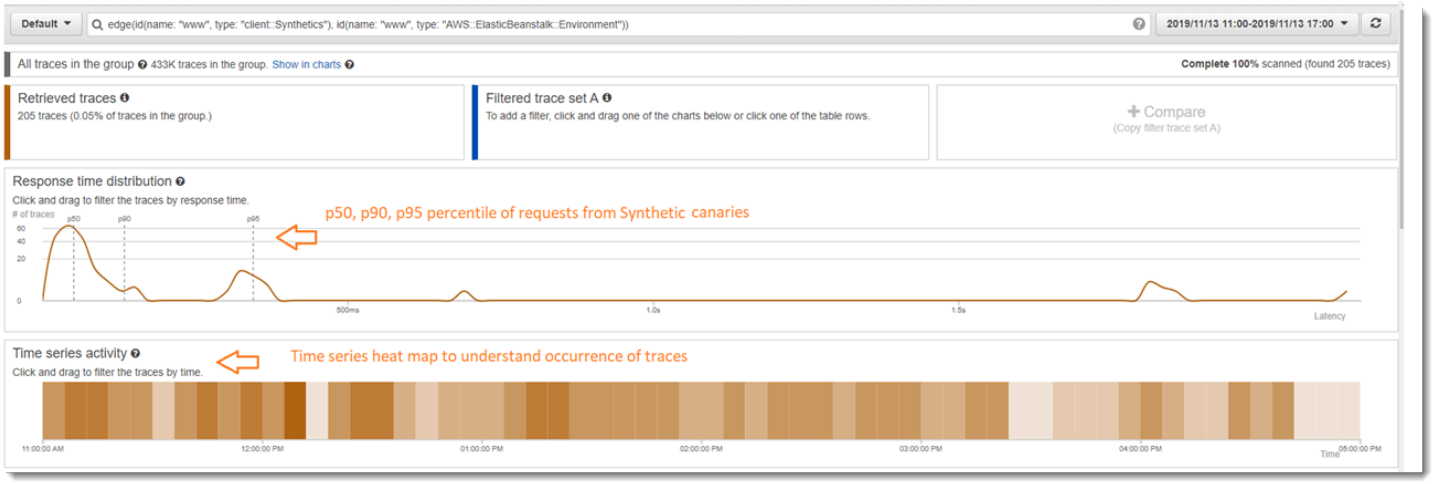

Canary
Map legend
Service details
Name: Canary
Type: client::Synthetics
View in Synthetics
Response distribution
Click and drag to select an area to zoom in on or use as a latency filter when viewing traces.
100%
80%
60%
40%
20%
0%
0
200ms
400ms
600ms
800ms
1.0s
1.2s
Duration
Response status
Choose response 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



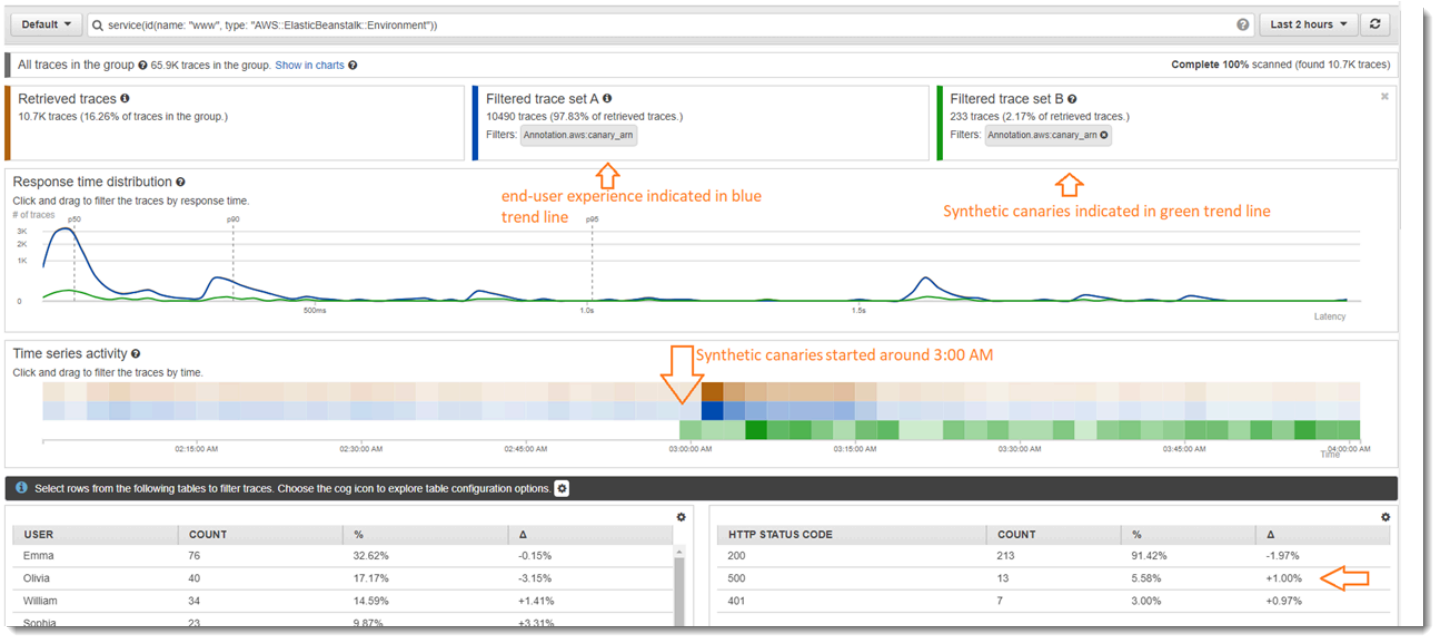
識別效能瓶頸和趨勢

您可以使用來自 Synthetics Canary 的連續流量，在一段時間內填入追蹤詳細資訊映射，來檢視端點效能隨時間的趨勢。



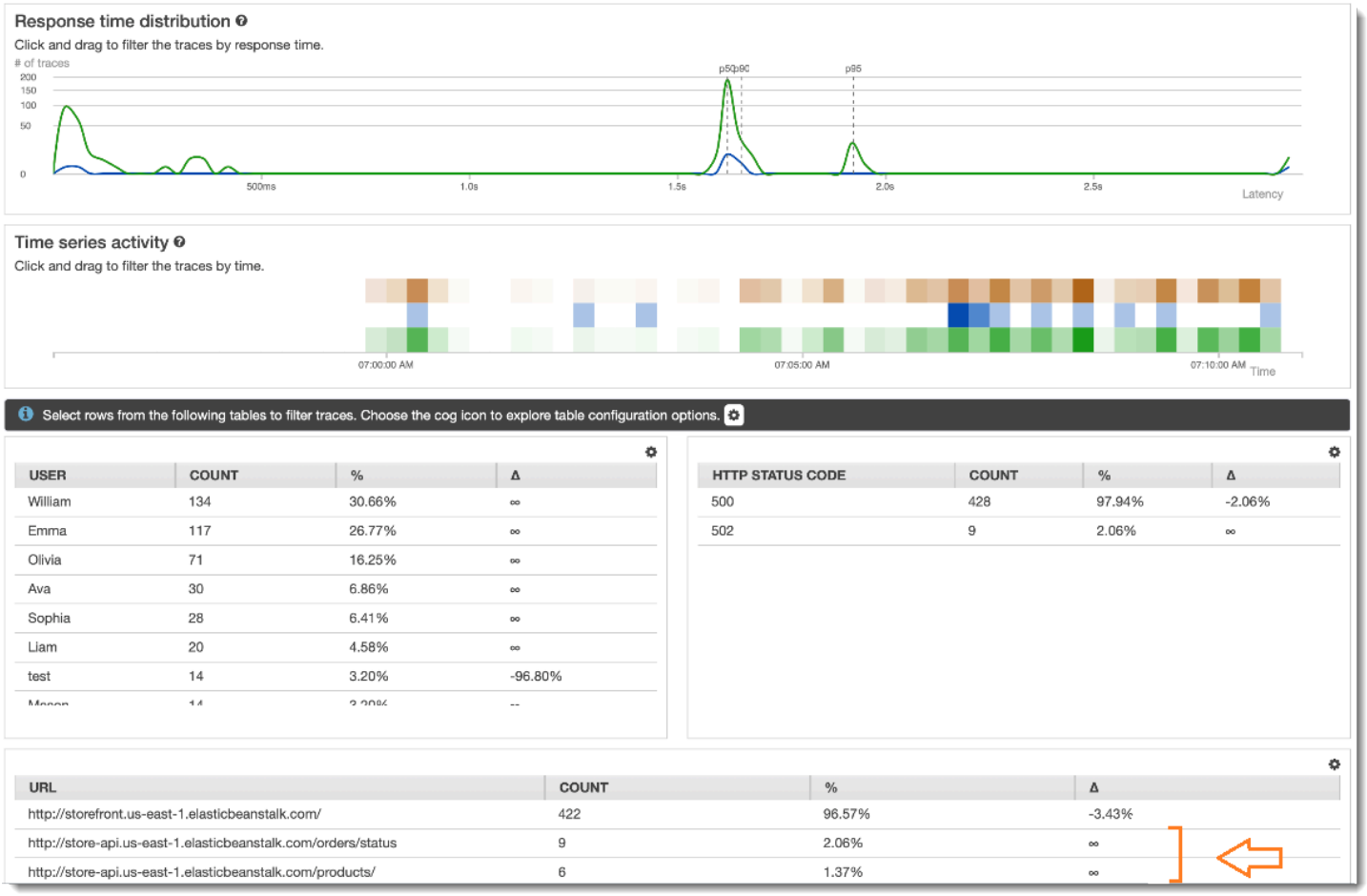
比較變更前後的延遲及錯誤率或容錯率

精確指出發生變更的時間，以將該變更與 Canary 所發現問題的增加相關聯。使用 X-Ray Analytics 主控台將時間範圍前後定義為不同的追蹤集，在回應時間分佈中建立視覺差異。



決定所有 API 和 URL 的必要 Canary 涵蓋範圍

利用 X-Ray Analytics 比較 Canary 和使用者的經歷。下面的 UI 中，藍色趨勢線表示 Canary，綠色趨勢線代表使用者。您也可以看到三個 URL 中有兩個沒有 Canary 測試。

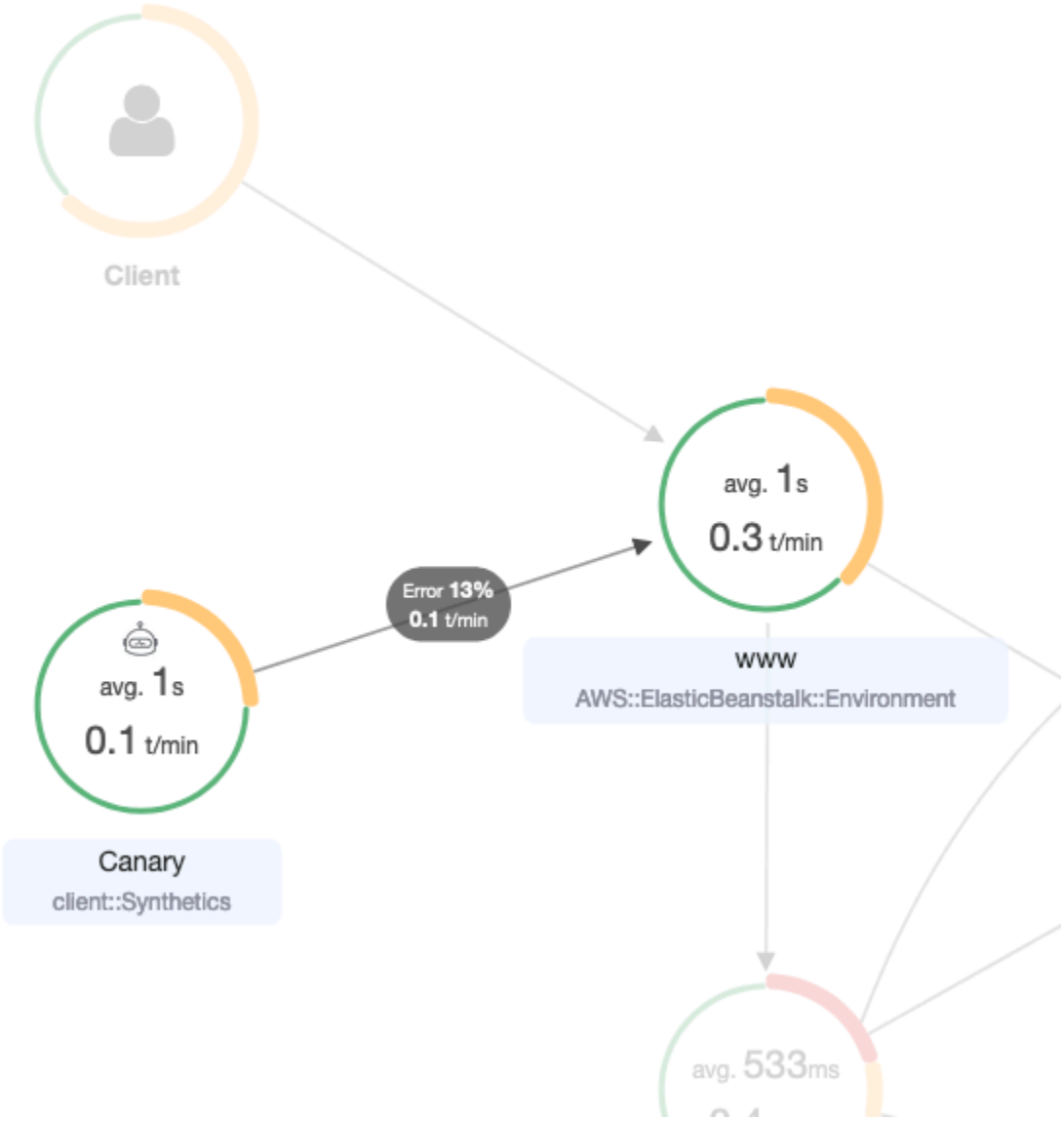


使用群組專注於 Synthetics 測試

您可以使用篩選條件表達式來建立 X-Ray 群組，以專注於特定的工作流程集，例如在 上執行之應用程式「www」的 Synthetics 測試 AWS Elastic Beanstalk。使用複雜的關鍵字 `service()`和 `edge()` 來篩選服務和邊緣。

Example 群組篩選條件表達式

```
"edge(id(name: "www", type: "client::Synthetics"), id(name: "www", type:
 "AWS::ElasticBeanstalk::Environment"))"
```



使用 追蹤 X-Ray 加密組態變更 AWS Config

AWS X-Ray 與 整合 AWS Config ，以記錄對 X-Ray 加密資源所做的組態變更。您可以使用 AWS Config 清查 X-Ray 加密資源、稽核 X-Ray 組態歷史記錄，並根據資源變更傳送通知。

AWS Config 支援將下列 X-Ray 加密資源變更記錄為事件：

- 組態變更 – 變更或新增加密金鑰，或還原至預設的 X-Ray 加密設定。

使用以下指示，了解如何在 X-Ray 和 之間建立基本連線 AWS Config。

建立 Lambda 函數觸發

您必須先擁有自訂 AWS Lambda 函數的 ARN，才能產生自訂 AWS Config 規則。按照這些指示，使用 Node.js 來建立基本函數，以根據 `XrayEncryptionConfig` 資源的狀態將合規或未合規的值傳回給 AWS Config 。

使用 AWS::XrayEncryptionConfig 變更觸發，建立 Lambda 函數

1. 開啟 Lambda 主控台。選擇 Create function (建立函數)。
2. 選擇 Blueprints (藍圖)，然後篩選藍圖程式庫的 config-rule-change-triggered 藍圖。按一下藍圖名稱中的連結，或選擇 Configure (設定) 以繼續。
3. 定義下列欄位以設定藍圖：
   - 針對 Name (名稱)，輸入名稱。
   - 在 Role (角色) 中，選擇 Create new role from template(s) (從範本建立新角色)。
   - 在 Role name (角色名稱) 中，輸入名稱。
   - 針對 Policy templates (政策範本)，選擇AWS Config Rules permissions (&CC; 規則許可)。
4. 選擇建立函數以在 AWS Lambda 主控台中建立和顯示您的函數。
5. 編輯您的函數程式碼，將 `AWS::EC2::Instance` 取代為 `AWS::XrayEncryptionConfig`。您也可以更新描述欄位，以反映這個變更。

   預設程式碼

```
    if (configurationItem.resourceType !== 'AWS::EC2::Instance') {
        return 'NOT_APPLICABLE';
    } else if (ruleParameters.desiredInstanceType ===
configurationItem.configuration.instanceType) {
        return 'COMPLIANT';
    }
        return 'NON_COMPLIANT';
```

   更新的程式碼

```
    if (configurationItem.resourceType !== 'AWS::XRay::EncryptionConfig') {
        return 'NOT_APPLICABLE';
    } else if (ruleParameters.desiredInstanceType ===
configurationItem.configuration.instanceType) {
        return 'COMPLIANT';
    }
```

```
        return 'NON_COMPLIANT';
```

6. 將以下內容新增至 IAM 中的執行角色，以存取 X-Ray。這些許可允許唯讀存取您的 X-Ray 資源。若未提供適當資源的存取權，則當評估與規則相關聯的 Lambda 函數 AWS Config 時，將導致 超出範圍的訊息。

```
{
    "Sid": "Stmt1529350291539",
    "Action": [
        "xray:GetEncryptionConfig"
    ],
    "Effect": "Allow",
    "Resource": "*"
 }
```

建立 X 射線的自訂 AWS Config 規則

建立 Lambda 函數時，請記下函數的 ARN，然後前往 AWS Config 主控台建立自訂規則。

建立 X-Ray 的 AWS Config 規則

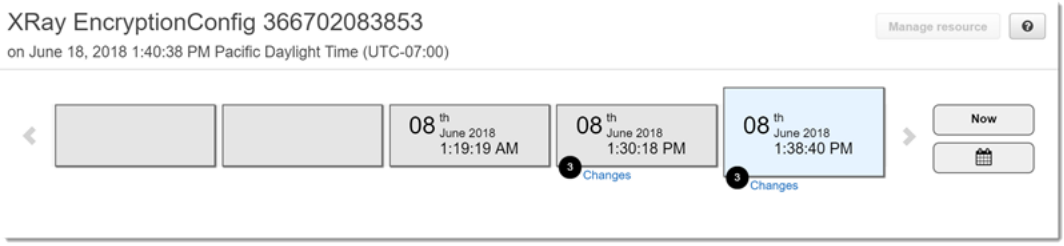
1. 開啟 AWS Config 主控台的規則頁面。
2. 選擇 Add rule (新增規則)，然後選擇 Add custom rule (新增自訂規則)。
3. 在AWS Lambda 函數 ARN 中，插入與您要使用的 Lambda 函數相關聯的 ARN。
4. 選擇要設定的觸發類型：
   - 組態變更 – 當符合規則範圍的任何資源在組態中變更時， AWS Config 觸發評估。評估會在 AWS Config 傳送組態項目變更通知後執行。
   - 定期 – 依您選擇的頻率 AWS Config 執行規則的評估 （例如，每 24 小時）。
5. 針對資源類型，在 X-Ray 區段EncryptionConfig中選擇 。
6. 選擇 Save (儲存)。

AWS Config 主控台會立即開始評估規則的合規。系統需要幾分鐘的時間來完成評估。

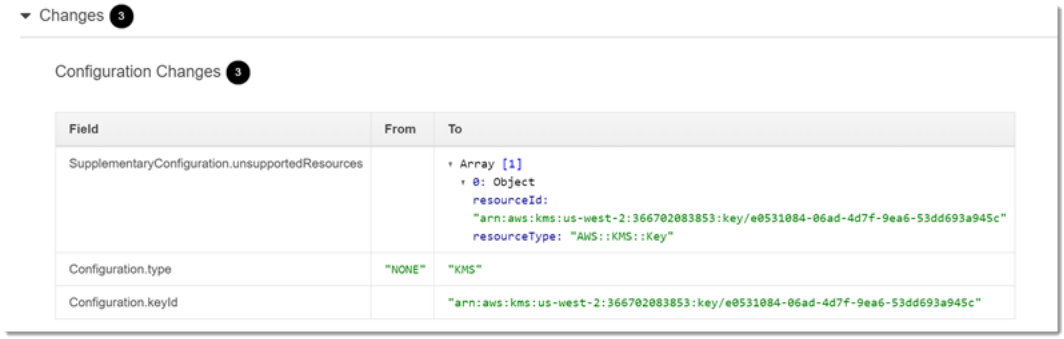
現在此規則符合規範， AWS Config 可以開始以時間軸的形式編譯稽核歷史記錄。 AWS Config 記錄資源變更。對於事件時間軸中的每個變更， 會以從/到格式 AWS Config 產生資料表，以顯示加密金鑰的 JSON 表示法中的變更。與 EncryptionConfig 相關聯的兩個欄位變更是 `Configuration.type` 和 `Configuration.keyID`。

範例結果

以下是 AWS Config 時間軸的範例，顯示在特定日期和時間所做的變更。



以下是 AWS Config 變更項目的範例。變更前/後格式可說明有哪些變更。此範例顯示預設 X-Ray 加密設定已變更為定義的加密金鑰。



Amazon SNS 通知

若要收到組態變更的通知，請將 AWS Config 設定為發佈 Amazon SNS 通知。如需詳細資訊，請參閱透過電子郵件監控 AWS Config 資源變更。

Amazon Elastic Compute Cloud 和 AWS X-Ray

您可以使用使用者資料指令碼在 Amazon EC2 執行個體上安裝和執行 X-Ray 協助程式。如需說明，請參閱 在 Amazon EC2 上執行 X-Ray 協助程式。

使用執行個體描述檔授予協助程式上傳追蹤資料的許可到 X-Ray。如需詳細資訊，請參閱授予協助程式將資料傳送至 X-Ray 的許可。

AWS Elastic Beanstalk 而且 AWS X-Ray

AWS Elastic Beanstalk 平台包括 X-Ray 協助程式。您可以在 Elastic Beanstalk 主控台或使用組態檔案設定 選項，以執行協助程式。

在 Java SE 平台中，您可以使用 Buildfile 檔案搭配執行個體上的 Maven 或 Gradle 來建置應用程式。適用於 Java 的 X-Ray 開發套件 適用於 Java 的 AWS SDK 可從 Maven 取得，因此您只能部署應用程式程式碼並建置執行個體，以避免綁定和上傳所有相依性。

您可以使用 Elastic Beanstalk 環境屬性來設定 X-Ray SDK。Elastic Beanstalk 用來將環境屬性傳遞給您應用程式的方法會因平台而異。根據您的平台使用 X-Ray 開發套件的環境變數或系統屬性。

- Node.js 平台 – 使用環境變數
- Java SE 平台 – 使用環境變數
- Tomcat 平台 – 使用系統屬性

如需詳細資訊，請參閱《 AWS Elastic Beanstalk 開發人員指南》中的設定 AWS X-Ray 偵錯。

Elastic Load Balancing 和 AWS X-Ray

Elastic Load Balancing 應用程式負載平衡器會將追蹤 ID 新增至名為 的標頭中的傳入 HTTP 請求`X-Amzn-Trace-Id`。

```
X-Amzn-Trace-Id: Root=1-5759e988-bd862e3fe1be46a994272793
```

X-Ray 追蹤 ID 格式

X-Ray `trace_id`由以連字號分隔的三個數字組成。例如：`1-58406520-a006649127e371903a2de979`。其中包含：

- 版本編號，即 1。
- 使用 8 個十六進位數字的 Unix epoch 時間原始請求的時間。

  例如，10：00AM 2016 年 12 月 1 日 PST 以 epoch `58406520` 為單位是`1480615200`秒或十六進位數字。
- 追蹤的全域唯一 96 位元識別符，以 24 十六進位數字表示。

負載平衡器不會將資料傳送至 X-Ray，也不會在服務地圖上顯示為節點。

如需詳細資訊，請參閱 Elastic Load Balancing 開發人員指南中的為您的 Application Load Balancer 請求追蹤。

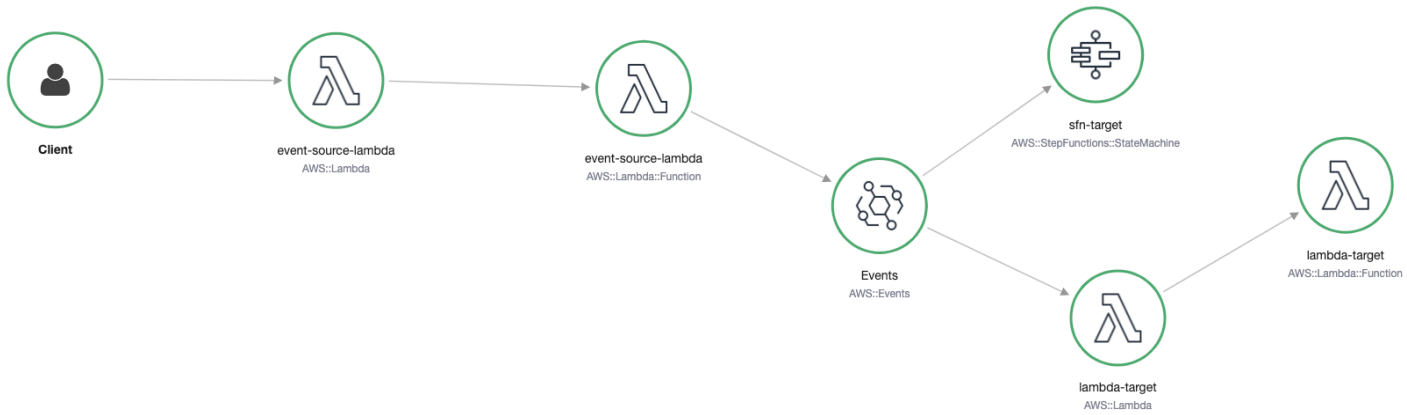
Amazon EventBridge 和 AWS X-Ray

AWS X-Ray 與 Amazon EventBridge 整合，以追蹤透過 EventBridge 傳遞的事件。如果使用 X-Ray SDK 檢測的服務將事件傳送至 EventBridge，追蹤內容會傳播到追蹤標頭中的下游事件目標。X-Ray SDK 會自動挑選追蹤標頭，並將其套用至任何後續的檢測。此連續性可讓使用者追蹤、分析和偵錯整個下游服務，並提供更完整的系統檢視。

如需詳細資訊，請參閱EventBridge 使用者指南》中的 EventBridge X-Ray 整合。 EventBridge

在 X-Ray 服務地圖上檢視來源和目標

X-Ray 追蹤映射會顯示連接來源和目標服務的 EventBridge 事件節點，如下列範例所示：



將追蹤內容傳播至事件目標

X-Ray SDK 可讓 EventBridge 事件來源將追蹤內容傳播至下游事件目標。下列特定語言範例示範從啟用主動追蹤的 Lambda 函數呼叫 EventBridge：

Java

新增 X-Ray 的必要相依性：

- AWS X-Ray 適用於 Java 的開發套件
- AWS X-Ray 適用於 Java 的 記錄器 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

將下列相依性新增至您的 requirements.txt 檔案：

```
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#

將下列 X-Ray 套件新增至您的 C# 相依性：

```
<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 而且 AWS X-Ray

您可以使用 AWS X-Ray 來追蹤 AWS Lambda 函數。Lambda 會執行 X-Ray 協助程式，並記錄區段，其中包含叫用和執行函數的詳細資訊。如需進一步檢測，您可以將 X-Ray 開發套件與 函數綁定，以記錄外撥通話並新增註釋和中繼資料。

如果您的 Lambda 函數是由另一個經檢測的服務呼叫，Lambda 會追蹤已經取樣的請求，而不需要任何其他組態。上游服務可以是經檢測的 Web 應用程式或其他 Lambda 函數。您的服務可以直接透過經檢測的 AWS SDK 用戶端叫用 函數，或使用經檢測的 HTTP 用戶端呼叫 API Gateway API。

AWS X-Ray 支援使用 AWS Lambda 和 Amazon SQS 追蹤事件驅動型應用程式。使用 CloudWatch 主控台查看與 Amazon SQS 佇列並由下游 Lambda 函數處理的每個請求的連線檢視。來自上游訊息生產者的追蹤會自動連結至來自下游 Lambda 消費者節點的追蹤，建立end-to-end檢視。如需詳細資訊，請參閱追蹤事件驅動的應用程式。

> **Note**
>
> 如果您已啟用下游 Lambda 函數的追蹤，您還必須為呼叫下游函數的根 Lambda 函數啟用追蹤，下游函數才能產生追蹤。

如果您的 Lambda 函數按排程執行，或是由未檢測的服務叫用，您可以設定 Lambda 以使用主動追蹤來取樣和記錄叫用。

在 AWS Lambda 函數上設定 X-Ray 整合

1. 開啟 AWS Lambda 主控台。
2. 從左側導覽列選取函數。
3. 選擇函數。
4. 在組態索引標籤上，向下捲動至其他監控工具卡。您也可以選取左側導覽窗格上的監控和操作工具來尋找此卡片。
5. 選擇 Edit (編輯)。
6. 在 AWS X-Ray 下，啟用 Active tracing (主動追蹤)。

在具有對應 X-Ray 開發套件的執行時間上，Lambda 也會執行 X-Ray 協助程式。

Lambda 上的 X-Ray SDKs

- 適用於 Go 的 X-Ray 開發套件 – Go 1.7 和更新版本執行時間
- 適用於 Java 的 X-Ray 開發套件 – Java 8 執行時間
- 適用於 Node.js 的 X-Ray 開發套件 – Node.js 4.3 和更新版本的執行時間
- 適用於 Python 的 X-Ray SDK – Python 2.7、Python 3.6 和更新版本的執行時間

- 適用於 .NET 的 X-Ray 開發套件 – .NET Core 2.0 和更新版本的執行時間

若要在 Lambda 上使用 X-Ray 開發套件，請在每次建立新版本時將其與函數程式碼綁定。您可以使用您用來檢測在其他 服務上執行之應用程式的相同方法，來檢測 Lambda 函數。主要差別是，您無法使用軟體開發套件來檢測傳入的請求、制定抽樣決策及建立區段。

檢測 Lambda 函數和 Web 應用程式之間的另一個區別是，函數程式碼無法修改 Lambda 建立並傳送至 X-Ray 的區段。您可以建立子區段並記錄註釋和中繼資料，但您無法新增註釋和中繼資料到父區段。

如需詳細資訊，請參閱《 AWS Lambda 開發人員指南》中的使用 AWS X-Ray。

Amazon SNS 和 AWS X-Ray

您可以使用 AWS X-Ray 搭配 Amazon Simple Notification Service (Amazon SNS)，在請求通過 SNS 主題到 SNS 支援的訂閱服務時追蹤和分析請求。使用 X-Ray 追蹤搭配 Amazon SNS 來分析訊息及其後端服務中的延遲，例如請求在主題中花費的時間，以及將訊息傳遞給每個主題訂閱所需的時間。Amazon SNS 支援標準和 FIFO 主題的 X-Ray 追蹤。

如果您從已使用 X-Ray 檢測的服務發佈至 Amazon SNS 主題，Amazon SNS 會將追蹤內容從發佈者傳遞給訂閱者。此外，您可以開啟主動追蹤，將 Amazon SNS 訂閱的區段資料傳送至 X-Ray，以取得從經檢測的 SNS 用戶端發佈的訊息。使用 Amazon SNS 主控台或使用 Amazon SNS API 或 CLI 開啟 Amazon SNS 主題的主動追蹤。如需檢測 SNS 用戶端的詳細資訊，請參閱檢測您的應用程式。

設定 Amazon SNS 主動追蹤

您可以使用 Amazon SNS 主控台或 AWS CLI 或 SDK 來設定 Amazon SNS 主動追蹤。

當您使用 Amazon SNS 主控台時，Amazon SNS 會嘗試為 SNS 建立呼叫 X-Ray 的必要許可。如果您沒有足夠的許可來修改 X-Ray 資源政策，則可以拒絕嘗試。如需這些許可的詳細資訊，請參閱《Amazon Simple Notification Service 開發人員指南》中的 Amazon SNS 中的身分和存取管理，以及 Amazon SNS 存取控制的範例案例。如需使用 Amazon SNS 主控台開啟主動追蹤的詳細資訊，請參閱《Amazon Simple Notification Service 開發人員指南》中的在 Amazon SNS 主題上啟用主動追蹤。

使用 AWS CLI 或 SDK 開啟主動追蹤時，您必須使用以資源為基礎的政策手動設定許可。使用 `PutResourcePolicy` 設定具有必要資源型政策的 X-Ray，以允許 Amazon SNS 將追蹤傳送至 X-Ray。

Example Amazon SNS 主動追蹤的 X-Ray 資源型政策範例

此範例政策文件指定 Amazon SNS 將追蹤資料傳送至 X-Ray 所需的許可：

```
{
    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"
          }
        }
      }
    ]
  }
```

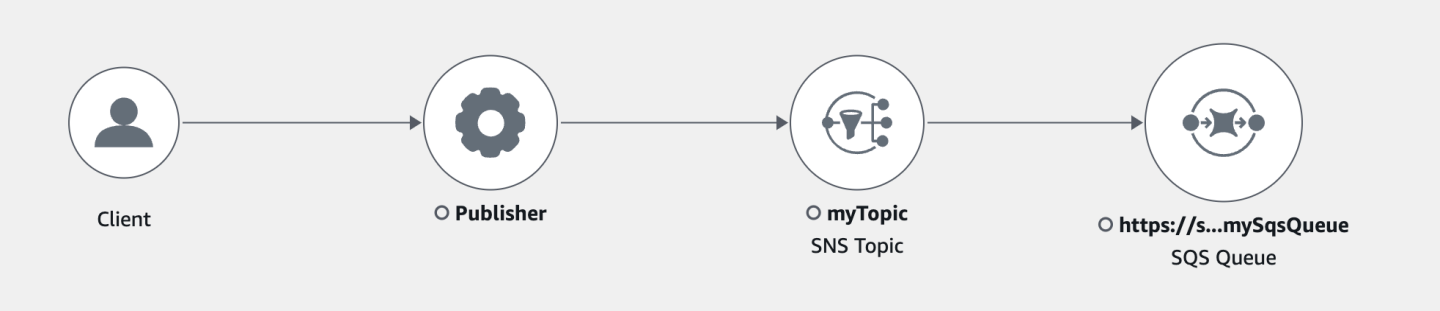
使用 CLI 建立資源型政策，提供 Amazon SNS 將追蹤資料傳送至 X-Ray 的許可：

```
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" } } } ] }'
```

若要使用這些範例，請將 *partition*、*account-id*、 *region*和 取代*topic-name*為您的特定 AWS 分割區、區域、帳戶 ID 和 Amazon SNS 主題名稱。若要授予所有 Amazon SNS 主題將追蹤資料傳送至 X-Ray 的許可，請將主題名稱取代為 *。

在 X-Ray 主控台中檢視 Amazon SNS 發佈者和訂閱者追蹤

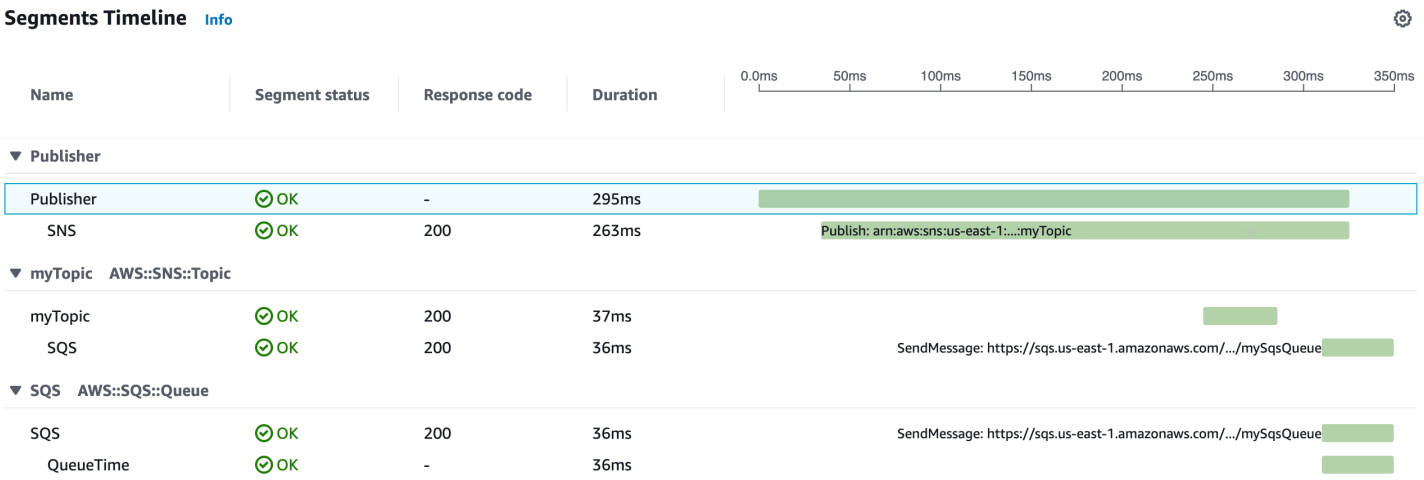
使用 X-Ray 主控台檢視追蹤地圖和追蹤詳細資訊，以顯示 Amazon SNS 發佈者和訂閱者的連線檢視。為主題開啟 Amazon SNS 主動追蹤時，X-Ray 追蹤映射和追蹤詳細資訊映射會顯示 Amazon SNS 發佈者、Amazon SNS 主題和下游訂閱者的連線節點：



選擇跨越 Amazon SNS 發佈者和訂閱者的追蹤之後，X-Ray 追蹤詳細資訊頁面會顯示追蹤詳細資訊地圖和客群時間軸。

Example Amazon SNS 發佈者和訂閱者的時間表範例

此範例顯示的時間軸包含 Amazon SNS 發佈者，該發佈者會將訊息傳送至 Amazon SNS 主題，該主題由 Amazon SQS 訂閱者處理。

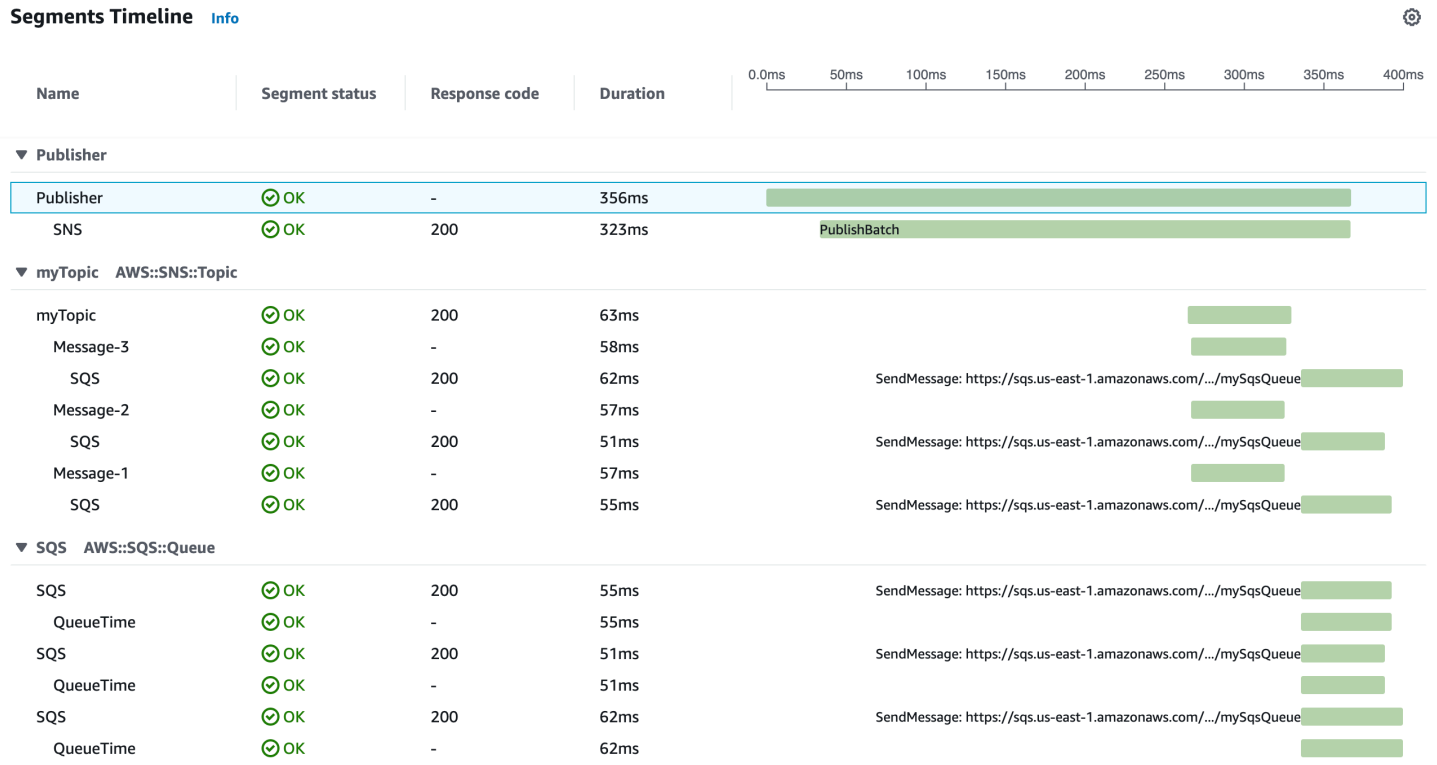


上述時間軸範例提供有關 Amazon SNS 訊息流程的詳細資訊：

- SNS 區段代表從用戶端進行 `Publish` API 呼叫的往返持續時間。
- myTopic 區段代表發佈請求的 Amazon SNS 回應延遲。
- SQS 子區段代表 Amazon SNS 將訊息發佈至 Amazon SQS 佇列所需的往返時間。
- myTopic 區段與 SQS 子區段之間的時間代表訊息在 Amazon SNS 系統中花費的時間。

Example 批次處理 Amazon SNS 訊息的時間軸範例

如果在單一追蹤中批次處理多個 Amazon SNS 訊息，區段時間軸會顯示代表每個已處理訊息的區段。



AWS Step Functions 而且 AWS X-Ray

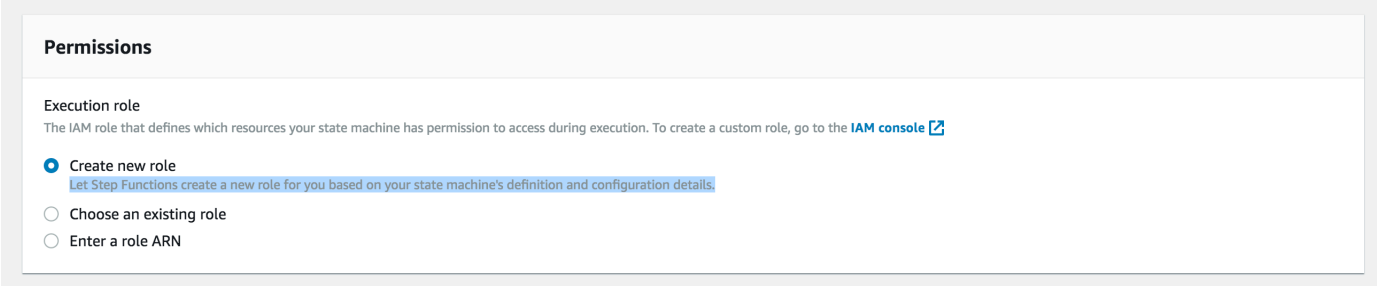
AWS X-Ray 與 整合 AWS Step Functions ，以追蹤和分析 Step Functions 的請求。您可以視覺化狀態機器的元件、識別效能瓶頸，以及對導致錯誤的請求進行故障診斷。如需詳細資訊，請參閱《 AWS Step Functions 開發人員指南》中的 AWS X-Ray 和 Step Functions。

在建立新的狀態機器時啟用 X-Ray 追蹤

1. 開啟 Step Functions 主控台，網址為 https://console.aws.amazon.com/states/。
2. 選擇建立狀態機器。
3. 在定義狀態機器頁面上，選擇使用程式碼片段撰寫或開始使用範本。如果您選擇執行範例專案，則無法在建立期間啟用 X-Ray 追蹤。反之，請在建立狀態機器後啟用 X-Ray 追蹤。
4. 選擇 Next (下一步)。
5. 在指定詳細資訊頁面上，設定您的狀態機器。
6. 選擇啟用 X-Ray 追蹤。

在現有狀態機器中啟用 X-Ray 追蹤

1. 在 Step Functions 主控台中，選取您要啟用追蹤的狀態機器。
2. 選擇編輯。
3. 選擇啟用 X-Ray 追蹤。
4. （選用）透過從許可視窗中選擇建立新角色，為您的狀態機器自動產生新角色以包含 X-Ray 許可。



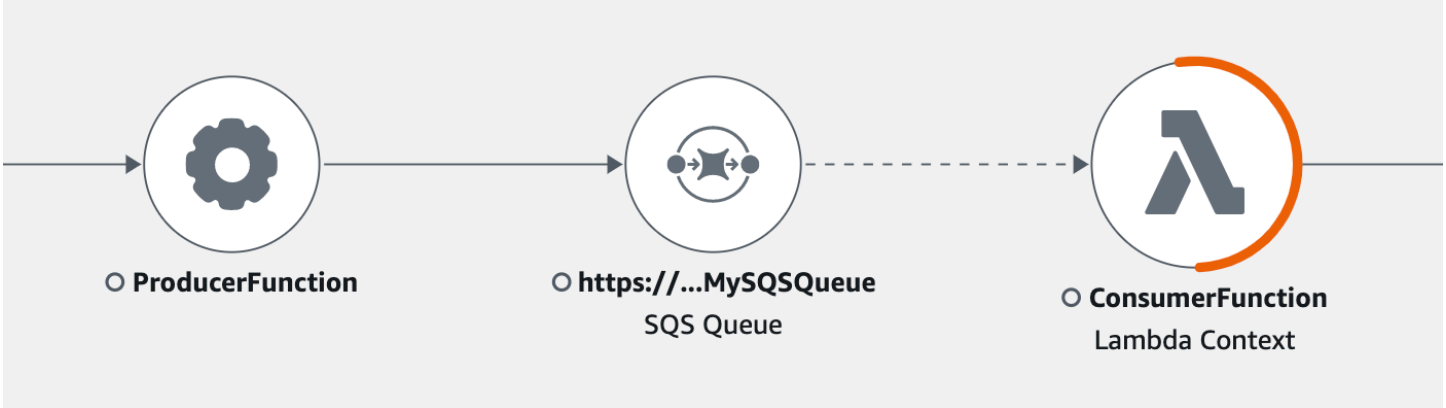
5. 選擇 Save (儲存)。

> **Note**
>
> 當您建立新的狀態機器時，如果對請求進行取樣並在 Amazon API Gateway 或 等上游服務中啟用追蹤，則會自動追蹤 AWS Lambda。對於未透過主控台設定的任何現有狀態機器，例如透過 AWS CloudFormation 範本，請檢查您是否具有授予足夠許可以啟用 X-Ray 追蹤的 IAM 政策。

Amazon SQS 和 AWS X-Ray

AWS X-Ray 與 Amazon Simple Queue Service (Amazon SQS) 整合，以追蹤透過 Amazon SQS 佇列傳遞的訊息。如果服務使用 X-Ray SDK 追蹤請求，Amazon SQS 可以傳送追蹤標頭，並繼續使用一致的追蹤 ID 將原始追蹤從寄件者傳播給消費者。追蹤連續性可讓使用者追蹤、分析和偵錯整個下游服務。

AWS X-Ray 支援使用 Amazon SQS 和 追蹤事件驅動型應用程式 AWS Lambda。使用 CloudWatch 主控台查看與 Amazon SQS 佇列並由下游 Lambda 函數處理之每個請求的連線檢視。來自上游訊息生產者的追蹤會自動連結至來自下游 Lambda 消費者節點的追蹤，建立end-to-end檢視。如需詳細資訊，請參閱追蹤事件驅動的應用程式。



Amazon SQS 支援下列追蹤標頭檢測：

- 預設 HTTP 標頭 – 當您透過 SDK 呼叫 Amazon SQS 時，X-Ray AWS SDK 會自動將追蹤標頭填入為 HTTP 標頭。預設追蹤標頭由 `X-Amzn-Trace-Id` 攜帶，並對應至 `SendMessage` 或 `SendMessageBatch` 請求中包含的所有訊息。若要進一步了解預設 HTTP 標頭，請參閱 追蹤標頭。
- **`AWSTraceHeader`** 系統屬性 – `AWSTraceHeader`是 Amazon SQS 預留的訊息系統屬性，用於在佇列中攜帶 X-Ray 追蹤標頭與訊息。即使未透過 X-Ray SDK 自動檢測，`AWSTraceHeader`也可以使用，例如在為新語言建置追蹤 SDK 時。同時設定這兩個標頭檢測時，訊息系統屬性會覆寫 HTTP 追蹤標頭。

在 Amazon EC2 上執行時，Amazon SQS 支援一次處理一則訊息。這適用於在內部部署主機上執行，以及使用容器服務時 AWS Fargate，例如 Amazon ECS 或 AWS App Mesh。

追蹤標頭會從 Amazon SQS 訊息大小和訊息屬性配額中排除。啟用 X-Ray 追蹤不會超過您的 Amazon SQS 配額。若要進一步了解 AWS 配額，請參閱 Amazon SQS Quotas。

傳送 HTTP 追蹤標頭

Amazon SQS 中的寄件者元件可以透過 `SendMessageBatch`或 `SendMessage`呼叫自動傳送追蹤標頭。檢測 AWS SDK 用戶端時，可以透過 X-Ray SDK 支援的所有語言自動追蹤它們。您在這些服務中存取的追蹤 AWS 服務 和資源 （例如 Amazon S3 儲存貯體或 Amazon SQS 佇列），會在 X-Ray 主控台的追蹤地圖上顯示為下游節點。

若要了解如何使用您偏好的語言追蹤 AWS SDK 呼叫，請參閱支援的 SDKs 中的下列主題：

- Go – 使用適用於 Go 的 X-Ray AWS 開發套件追蹤 SDK 呼叫
- Java – 使用適用於 Java 的 X-Ray AWS 開發套件追蹤 SDK 呼叫

- Node.js – 使用適用於 Node.js 的 X-Ray AWS 開發套件追蹤 SDK 呼叫
- Python – 使用適用於 Python 的 X-Ray AWS 開發套件追蹤 SDK 呼叫
- Ruby – 使用適用於 Ruby 的 X-Ray AWS 開發套件追蹤 SDK 呼叫
- .NET – 使用適用於 AWS .NET 的 X-Ray 開發套件追蹤 SDK 呼叫

擷取追蹤標頭和復原追蹤內容

如果您使用的是 Lambda 下游消費者，則會自動進行追蹤內容傳播。若要繼續與其他 Amazon SQS 取用者進行內容傳播，您必須手動檢測接收器元件的交接。

復原追蹤內容有三個主要步驟：

- 呼叫 `ReceiveMessage` API，從 `AWSTraceHeader` 屬性的佇列接收訊息。
- 從屬性擷取追蹤標頭。
- 從標頭復原追蹤 ID。選擇性地將更多指標新增至區段。

以下是使用適用於 Java 的 X-Ray 開發套件撰寫的範例實作。

Example ：擷取追蹤標頭和復原追蹤內容

```
// 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));
```

```
    }
}
```

Amazon S3 和 AWS X-Ray

AWS X-Ray 與 Amazon S3 整合，以追蹤上游請求，以更新應用程式的 S3 儲存貯體。如果服務使用 X-Ray SDK 追蹤請求，Amazon S3 可以將追蹤標頭傳送給下游事件訂閱者，例如 AWS Lambda、Amazon SQS 和 Amazon SNS。X-Ray 可啟用 Amazon S3 事件通知的追蹤訊息功能。

您可以使用 X-Ray 追蹤圖來檢視 Amazon S3 與應用程式所使用其他服務之間的連線。您也可以使用主控台來檢視指標，例如平均延遲和失敗率。如需 X-Ray 主控台的詳細資訊，請參閱 使用 X-Ray 主控台。

Amazon S3 支援預設 http 標頭檢測。當您透過 SDK 呼叫 Amazon S3 時，X-Ray AWS SDK 會自動將追蹤標頭填入為 HTTP 標頭。預設追蹤標頭由 承載`X-Amzn-Trace-Id`。若要進一步了解追蹤標頭，請參閱 概念頁面上追蹤標頭的 。Amazon S3 追蹤內容傳播支援下列訂閱者：Lambda、SQS 和 SNS。由於 SQS 和 SNS 本身不會發出區段資料，因此它們不會在 S3 觸發時出現在追蹤或追蹤映射中，即使它們會將追蹤標頭傳播到下游服務。

設定 Amazon S3 事件通知

使用 Amazon S3 通知功能，當儲存貯體中發生特定事件時，您會收到通知。然後，這些通知可以傳播到應用程式中的下列目的地：

- Amazon Simple Notification Service (Amazon SNS)
- Amazon Simple Queue Service (Amazon SQS)
- AWS Lambda

如需支援的事件清單，請參閱《Amazon S3 開發人員指南》中的支援的事件類型。

Amazon SNS 和 Amazon SQS

若要發佈通知至 SNS 主題或 SQS 佇列，您必須先授予 Amazon S3 許可。若要授予這些許可，請將 AWS Identity and Access Management (IAM) 政策連接至目的地 SNS 主題或 SQS 佇列。若要進一步了解所需的 IAM 政策，請參閱授予發佈訊息至 SNS 主題或 SQS 佇列的許可。

如需整合 SNS 和 SQS 與 X-Ray 的相關資訊，請參閱 Amazon SNS 和 AWS X-Ray和 Amazon SQS 和 AWS X-Ray。

AWS Lambda

當您使用 Amazon S3 主控台為 Lambda 函數設定 S3 儲存貯體的事件通知時，主控台會設定 Lambda 函數的必要許可，讓 Amazon S3 具有從儲存貯體叫用函數的許可。如需詳細資訊，請參閱《Amazon Simple Storage Service 主控台使用者指南》中的如何啟用和設定 S3 儲存貯體的事件通知？。

您也可以從 授予 Amazon S3 許可 AWS Lambda ，以叫用 Lambda 函數。如需詳細資訊，請參閱《AWS Lambda 開發人員指南》中的教學課程：搭配 Amazon S3 使用 AWS Lambda。

如需整合 Lambda 與 X-Ray 的詳細資訊，請參閱在 AWS Lambda 中檢測 Java 程式碼。

檢測您的應用程式 AWS X-Ray

檢測您的應用程式涉及傳送傳入和傳出請求的追蹤資料，以及應用程式中的其他事件，以及每個請求的中繼資料。您可以根據您的特定需求，選擇或結合多種不同的檢測選項：

- 自動檢測 – 以零程式碼變更檢測您的應用程式，通常透過組態變更、新增自動檢測代理程式或其他機制。
- 程式庫檢測 – 進行最少的應用程式程式碼變更，以新增以特定程式庫或架構為目標的預先建置檢測，例如 AWS SDK、Apache HTTP 用戶端或 SQL 用戶端。
- 手動檢測 – 將檢測程式碼新增至您要傳送追蹤資訊的每個位置的應用程式。

有數個 SDKs、代理程式和工具可用來檢測您的應用程式以進行 X-Ray 追蹤。

主題

使用 AWS Distro for OpenTelemetry 檢測您的應用程式

AWS Distro for OpenTelemetry (ADOT) 是基於雲端原生運算基金會 (CNCF) OpenTelemetry 專案的 AWS 分佈。OpenTelemetry 提供一組開放原始碼 APIs、程式庫和代理程式，以收集分散式追蹤和指標。此工具組是上游 OpenTelemetry 元件的分佈，包括 SDKs、自動檢測代理程式和 測試、最佳化、保護和支援的收集器 AWS。

透過 ADOT，工程師可以一次檢測其應用程式，並將相關指標和追蹤傳送至多個 AWS 監控解決方案 AWS X-Ray，包括 Amazon CloudWatch 和 Amazon OpenSearch Service。

將 X-Ray 與 ADOT 搭配使用需要兩個元件：啟用 OpenTelemetry SDK 以與 X-Ray 搭配使用，以及啟用 AWS Distro for OpenTelemetry Collector 以與 X-Ray 搭配使用。如需搭配 AWS AWS X-Ray 和其他 使用 Distro for OpenTelemetry 的詳細資訊 AWS 服務，請參閱 AWS Distro for OpenTelemetry 文件。

如需語言支援和用量的詳細資訊，請參閱 AWS GitHub 上的可觀測性。

> **Note**
>
> 您現在可以使用 CloudWatch 代理程式從 Amazon EC2 執行個體和內部部署伺服器收集指標、日誌和追蹤。CloudWatch 代理程式 1.300025.0 版及更新版本可以從 OpenTelemetry 或 X-Ray 用戶端 SDKs 收集追蹤，並將其傳送至 X-Ray。使用 CloudWatch 代理程式而非 AWS Distro for OpenTelemetry (ADOT) Collector 或 X-Ray 協助程式收集追蹤，可協助您減少管理的代理程式數量。如需詳細資訊，請參閱《CloudWatch 使用者指南》中的 CloudWatch 代理程式主題。 CloudWatch

ADOT 包括下列項目：

- AWS Distro for OpenTelemetry Go
- AWS Distro for OpenTelemetry Java
- AWS Distro for OpenTelemetry JavaScript
- AWS Distro for OpenTelemetry Python
- AWS Distro for OpenTelemetry .NET

ADOT 目前包含 Java 和 Python 的自動檢測支援。此外，ADOT 可透過 ADOT Managed AWS Lambda Layers，使用 Java、Node.js 和 Python 執行時間自動檢測 Lambda 函數及其下游請求。https://aws-otel.github.io/docs/getting-started/lambda

適用於 Java 和 Go SDKs支援 X-Ray 集中式取樣規則。如果您需要支援其他語言的 X-Ray 取樣規則，請考慮使用 AWS X-Ray SDK。

> **Note**
>
> 您現在可以將 W3C IDs 傳送至 X-Ray。根據預設，使用 OpenTelemetry 建立的追蹤具有以 W3C 追蹤內容規格為基礎的追蹤 ID 格式。這與使用 X-Ray SDK 或整合 X-Ray AWS 的服務所建立IDs 格式不同。若要確保 X-Ray 接受 W3C 格式IDs，您必須使用 AWS X-Ray Exporter 0.86.0 版或更新版本，其中包含 ADOT Collector 0.34.0 版和更新版本。匯出工具的舊版會驗證追蹤 ID 時間戳記，這可能會導致 W3C IDs 遭拒。

使用 AWS X-Ray SDKs 檢測您的應用程式

AWS X-Ray 包含一組語言特定的 SDKs用於檢測您的應用程式以將追蹤傳送至 X-Ray。每個 X-Ray 開發套件都提供下列項目：

- 攔截程式，可新增至您的程式碼以追蹤傳入的 HTTP 請求
- 用戶端處理常式可檢測您的應用程式用來呼叫其他 的 AWS SDK 用戶端 AWS 服務
- HTTP 用戶端，用於檢測對其他內部和外部 HTTP Web 服務的呼叫

X-Ray SDKs也支援檢測對 SQL 資料庫的呼叫、自動 AWS SDK 用戶端檢測和其他功能。SDK 不會直接將追蹤資料傳送至 X-Ray，而是將 JSON 區段文件傳送至監聽 UDP 流量的協助程式程序。X-Ray 協助程式會緩衝佇列中的區段，並批次上傳至 X-Ray。

提供下列語言特定的 SDKs：

- AWS X-Ray 適用於 Go 的 SDK
- AWS X-Ray 適用於 Java 的開發套件
- AWS X-Ray 適用於 Node.js 的 SDK
- AWS X-Ray 適用於 Python 的 SDK
- AWS X-Ray 適用於 .NET 的 SDK
- AWS X-Ray 適用於 Ruby 的 SDK

X-Ray 目前包含 Java 的自動檢測支援。

選擇 AWS Distro for OpenTelemetry 和 X-Ray SDKs

X-Ray 隨附的SDKs是 提供的緊密整合檢測解決方案的一部分 AWS。 AWS Distro for OpenTelemetry 是更廣泛的產業解決方案的一部分，其中 X-Ray 只是許多追蹤解決方案之一。您可以使用任一方法在 X-Ray 中實作end-to-end追蹤，但請務必了解差異，以判斷對您最有用的方法。

如果您需要下列項目，建議您使用 AWS Distro for OpenTelemetry 檢測應用程式：

- 能夠將追蹤傳送至多個不同的追蹤後端，而無需重新檢測程式碼
- 支援 OpenTelemetry 社群維護的每種語言的大量程式庫檢測
- 完全受管的 Lambda 層，可封裝收集遙測資料所需的一切，而無需在使用 Java、Python 或 Node.js 時變更程式碼

> **Note**
>
> AWS Distro for OpenTelemetry 提供更簡單的 Lambda 函數檢測入門體驗。不過，由於 OpenTelemetry 提供的靈活性，您的 Lambda 函數將需要額外的記憶體，而且調用可能會遇到冷啟動延遲增加，這可能會導致額外費用。如果您要最佳化低延遲，且不需要 OpenTelemetry 的進階功能，例如可動態設定的後端目的地，您可能想要使用 AWS X-Ray 開發套件來檢測應用程式。

如果您需要下列項目，建議您選擇 X-Ray 開發套件來檢測您的應用程式：

- 緊密整合的單一供應商解決方案
- 與 X-Ray 集中式取樣規則整合，包括能夠從 X-Ray 主控台設定取樣規則，並在使用 Node.js、Python、Ruby 或 .NET 時自動跨多個主機使用它們

交易搜尋

交易搜尋是一種互動式分析體驗，可用來完整了解應用程式交易範圍。跨度是分散式追蹤中的基本操作單位，代表應用程式或系統中的特定動作或任務。每個範圍都會記錄交易特定區段的詳細資訊。這些詳細資訊包括開始和結束時間、持續時間和相關聯的中繼資料，其中可能包括業務屬性，例如客戶 IDs和訂單 IDs。跨度會排列在父子階層中。此階層會形成完整的追蹤，映射不同元件或服務的交易流程。

如需詳細資訊，請參閱交易搜尋。

OpenTelemetry Protocol (OTLP) 端點

OpenTelemetry 是一種開放原始碼可觀測性架構，可為 IT 團隊提供標準化通訊協定和工具，用於收集和路由遙測資料。它提供統一格式，用於檢測、產生、收集和匯出應用程式遙測資料，例如指標、日誌和追蹤，以監控平台進行分析和洞察。透過使用 OpenTelemetry，團隊可以避免廠商鎖定，確保其可觀測性解決方案的彈性。

您可以使用 OpenTelemetry 將追蹤直接傳送至 OpenTelemetry Protocol (OTLP) 端點，並在 CloudWatch Application Signals out-of-the 中取得立即可用的應用程式效能監控體驗。

如需詳細資訊，請參閱 OpenTelemetry。

使用 Go

有兩種方式可以檢測 Go 應用程式，以將追蹤傳送至 X-Ray：

- AWS Distro for OpenTelemetry Go – 透過 AWS Distro for OpenTelemetry Collector， AWS 提供一組開放原始碼程式庫，用於將相關指標和追蹤傳送至多個 AWS 監控解決方案，包括 Amazon CloudWatch AWS X-Ray和 Amazon OpenSearch Service。
- AWS X-Ray SDK for Go – 一組程式庫，用於透過 X-Ray 協助程式產生追蹤並傳送至 X-Ray。

如需詳細資訊，請參閱選擇 AWS Distro for OpenTelemetry 和 X-Ray SDKs。

AWS Distro for OpenTelemetry Go

使用 AWS Distro for OpenTelemetry Go，您可以檢測您的應用程式一次，並將相關指標和追蹤傳送至多個 AWS 監控解決方案 AWS X-Ray，包括 Amazon CloudWatch 和 Amazon OpenSearch Service。搭配 AWS Distro for OpenTelemetry 使用 X-Ray 需要兩個元件：啟用 OpenTelemetry SDK 以搭配 X-Ray 使用，以及啟用 AWS Distro for OpenTelemetry Collector 以搭配 X-Ray 使用。

若要開始使用，請參閱 AWS Distro for OpenTelemetry Go 文件。

如需搭配 AWS AWS X-Ray 和其他 使用 Distro for OpenTelemetry 的詳細資訊 AWS 服務，請參閱 AWS Distro for OpenTelemetry 或 AWS Distro for OpenTelemetry 文件。

如需語言支援和用量的詳細資訊，請參閱 AWS GitHub 上的可觀測性。

AWS X-Ray 適用於 Go 的 SDK

適用於 Go 的 X-Ray 開發套件是一組適用於 Go 應用程式的程式庫，提供產生追蹤資料並將其傳送至 X-Ray 協助程式的類別和方法。追蹤資料包含應用程式所提供服務之傳入 HTTP 請求的相關資訊，以及應用程式使用 AWS SDK、HTTP 用戶端或 SQL 資料庫連接器對下游服務進行的呼叫。您也可以手動建立區段，並將除錯資訊新增至註釋和中繼資料中。

使用 `go get` 以從軟體開發套件的 GitHub 儲存庫下載軟體開發套件：

```
$ go get -u github.com/aws/aws-xray-sdk-go/...
```

若是 Web 應用程式，請先使用 `xray.Handler` 函數來追蹤傳入請求。訊息處理常式會為每個追蹤的請求建立區段，並在傳送回應時完成區段。當區段開啟時，您可以使用軟體開發套件用戶端的方法，將資訊新增到區段，並建立子區段以追蹤下游呼叫。軟體開發套件也會在區段為開啟時自動記錄應用程式擲回的例外狀況。

對於由經檢測的應用程式或服務呼叫的 Lambda 函數，Lambda 會自動讀取追蹤標頭並追蹤取樣的請求。對於其他 函數，您可以設定 Lambda 來取樣和追蹤傳入的請求。無論哪種情況，Lambda 都會建立區段，並將其提供給 X-Ray 開發套件。

> **Note**
>
> 在 Lambda 上，X-Ray 開發套件是選用的。如果您未在函數中使用它，您的服務映射仍會包含 Lambda 服務的節點，以及每個 Lambda 函數的一個節點。透過新增 SDK，您可以檢測函數程式碼，將子區段新增至 Lambda 記錄的函數區段。如需更多資訊，請參閱AWS Lambda 而且 AWS X-Ray。

接著，請呼叫 AWS 函數以包裝您的用戶端。此步驟可確保 X-Ray 檢測到對任何用戶端方法的呼叫。您也可以檢測對 SQL 資料庫的呼叫。

開始使用 SDK 之後，請設定記錄器和中介軟體來自訂其行為。您可以新增外掛程式，以記錄執行應用程式所需的運算資源相關資料、定義抽樣規則以自訂抽樣行為，並設定日誌層級以在應用程式日誌中查看更多或更少的軟體開發套件資訊。

使用註釋與中繼資料，記錄應用程式所做的請求和工作等其他資訊。註釋是簡單的鍵/值對，系統會為其建立索引以用於篩選條件表達式，因此您可以搜尋包含特定資料的追蹤。中繼資料項目的限制性較低，可以記錄整個物件和陣列，任何可以序列化為 JSON 的項目。

> **標註與中繼資料**
>
> 註釋和中繼資料是您使用 X-Ray SDK 新增至區段的任意文字。註釋會編製索引，以便與篩選條件表達式搭配使用。中繼資料不會編製索引，但可以使用 X-Ray 主控台或 API 在原始區段中檢視。您授予 X-Ray 讀取存取權的任何人都可以檢視此資料。

當程式碼中有很多經過檢測的用戶端時，單一請求區段可能包含大量子區段，每個使用經檢測用戶端進行的呼叫都有一個子區段。您可以將用戶端呼叫包裝在自訂子區段中，以組織和群組子區段。您可以為整個函數或任何部分的程式碼建立自訂子區段，並記錄子區段上的中繼資料和註釋，而不必寫入父區段上的所有項目。

要求

適用於 Go 的 X-Ray 開發套件需要 Go 1.9 或更新版本。

軟體開發套件在編譯和執行時間需仰賴下列程式庫：

- AWS 適用於 Go 的 SDK 1.10.0 版或更新版本

軟體開發套件的 `README.md` 檔案中有宣告這些相依性。

參考文件

下載軟體開發套件之後，請本機建置和託管文件以在 Web 瀏覽器中檢視。

檢視參考文件

1. 導覽至 `$GOPATH/src/github.com/aws/aws-xray-sdk-go` (Linux 或 Mac) 目錄或 `%GOPATH%\src\github.com\aws\aws-xray-sdk-go` (Windows) 資料夾
2. 執行 `godoc` 命令。

   ```
   $ godoc -http=:6060
   ```

3. 在瀏覽器中開啟 `http://localhost:6060/pkg/github.com/aws/aws-xray-sdk-go/` 網址。

設定適用於 Go 的 X-Ray 的開發套件

您可以指定 X-Ray SDK for Go through environment 變數的組態、`Configure`使用 `Config` 物件呼叫 ，或假設預設值。環境變數優先於 `Config` 值，而後者則優先於任何預設值。

章節

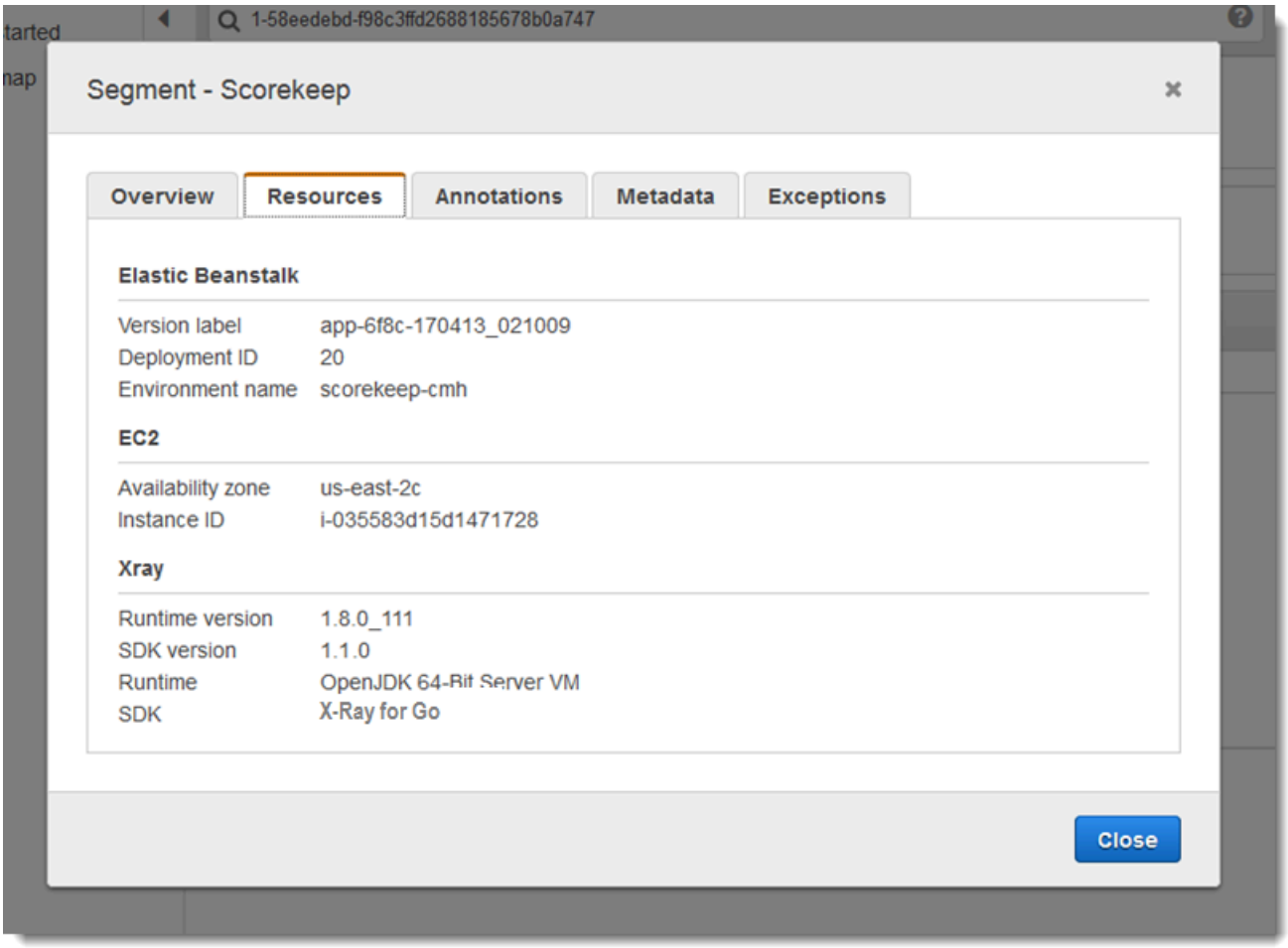
- 服務外掛程式
- 抽樣規則
- 日誌
- 環境變數
- 使用設定

服務外掛程式

使用 `plugins` 記錄託管您應用程式之服務的相關資訊。

外掛程式

- Amazon EC2 – `EC2Plugin` 新增執行個體 ID、可用區域和 CloudWatch Logs 群組。
- Elastic Beanstalk – `ElasticBeanstalkPlugin` 新增環境名稱、版本標籤和部署 ID。
- Amazon ECS – `ECSPlugin` 新增容器 ID。



若要使用外掛程式，請匯入以下其中一個套件。

```
"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"
```

每個外掛程式都有一個明確載入該外掛程式的 `Init()` 函數呼叫。

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",
  })
}
```

軟體開發套件也會使用外掛程式設定來設定區段上的 `origin` 欄位。這表示執行您應用程式 AWS 的資源類型。當您使用多個外掛程式時，開發套件會使用下列解析順序來判斷原始伺服器：ElasticBeanstalk > EKS > ECS > EC2。

抽樣規則

SDK 會使用您在 X-Ray 主控台中定義的抽樣規則來決定要記錄哪些請求。預設規則每秒追蹤第一個請求，以及所有傳送追蹤至 X-Ray 服務的任何其他請求的 5%。在 X-Ray 主控台中建立其他規則，以自訂為每個應用程式記錄的資料量。

軟體開發套件會依定義自訂規則的順序進行套用。如果請求符合多個自訂規則，軟體開發套件只會套用第一個規則。

> **Note**
>
> 如果開發套件無法達到 X-Ray 以取得取樣規則，它會每秒還原為第一個請求的預設本機規則，以及每個主機任何額外請求的 5%。如果主機沒有呼叫取樣 APIs許可，或無法連線至 X-Ray 協助程式，而該常駐程式可做為 SDK 進行 API 呼叫的 TCP 代理，則可能會發生這種情況。

您也可以設定 SDK 以從 JSON 文件載入取樣規則。開發套件可以使用本機規則作為 X-Ray 取樣不可用的備份，或僅使用本機規則。

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
  }
}
```

此範例會定義一個自訂規則和預設規則。自訂規則會套用 5% 的取樣率，沒有追蹤 下路徑的最低請求數/api/move/。預設規則會追蹤每秒的第一個請求和 10% 的額外請求。

在本機定義規則的缺點是，固定目標是由記錄器的每個執行個體獨立套用，而不是由 X-Ray 服務管理。當您部署更多主機時，固定速率會乘以，使得難以控制記錄的資料量。

在 上 AWS Lambda，您無法修改取樣率。如果您的函數是由 受檢測的服務呼叫，則 Lambda 會記錄產生該服務取樣請求的呼叫。如果啟用主動追蹤，且不存在追蹤標頭，Lambda 會做出抽樣決策。

若要提供備份規則，請使用 NewCentralizedStrategyWithFilePath 指向本機抽樣 JSON 檔案。

Example main.go – 本機取樣規則

```
s, _ := sampling.NewCentralizedStrategyWithFilePath("sampling.json") // path to local
 sampling json
xray.Configure(xray.Config{SamplingStrategy: s})
```

若僅要使用本機規則，請使用 NewLocalizedStrategyFromFilePath 指向本機抽樣 JSON 檔案。

Example main.go – 停用取樣

```
s, _ := sampling.NewLocalizedStrategyFromFilePath("sampling.json") // path to local
 sampling json
xray.Configure(xray.Config{SamplingStrategy: s})
```

日誌

> **Note**
>
> 從版本 1.0.0-rc.10 開始已取代 `xray.Config{}` 欄位 `LogLevel` 和 `LogFormat`。

X-Ray 使用下列界面進行記錄。在 `LogLevelInfo` 和以上時，預設記錄器會寫入 `stdout`。

```
type Logger interface {
 Log(level LogLevel, msg fmt.Stringer)
}

const (
 LogLevelDebug LogLevel = iota + 1
 LogLevelInfo
 LogLevelWarn
 LogLevelError
)
```

Example 寫入 **`io.Writer`**

```
xray.SetLogger(xraylog.NewDefaultLogger(os.Stderr, xraylog.LogLevelError))
```

環境變數

您可以使用環境變數來設定適用於 Go 的 X-Ray 開發套件。軟體開發套件支援以下變數。

- `AWS_XRAY_CONTEXT_MISSING` – 設定為 `RUNTIME_ERROR` 以在未開啟區段時，檢測程式碼嘗試記錄資料時擲回例外狀況。

  有效值

  - `RUNTIME_ERROR` – 捨棄執行時間例外狀況。
  - `LOG_ERROR` – 記錄錯誤並繼續 （預設）。

- IGNORE_ERROR – 忽略錯誤並繼續。

當您嘗試在未開啟請求時執行的啟動程式碼中，或在產生新執行緒的程式碼中，使用經檢測的用戶端時，可能會發生與缺少區段或子區段相關的錯誤。

- AWS_XRAY_TRACING_NAME – 設定 SDK 用於區段的服務名稱。
- AWS_XRAY_DAEMON_ADDRESS – 設定 X-Ray 協助程式接聽程式的主機和連接埠。根據預設，開發套件會將追蹤資料傳送至 127.0.0.1:2000。如果您已設定協助程式在不同的連接埠上接聽，或在不同的主機上執行，請使用此變數。
- AWS_XRAY_CONTEXT_MISSING – 設定 值，以判斷 SDK 如何處理遺漏的內容錯誤。若您嘗試在未開啟請求時於啟動程式碼中使用受檢測用戶端，或是在會產生新執行緒的程式碼中使用受檢測用戶端，就可能會發生與遺漏區段或子區段相關的錯誤。
  - RUNTIME_ERROR – 根據預設，軟體開發套件會設定為擲回執行時間例外狀況。
  - LOG_ERROR – 設定 以記錄錯誤並繼續。

環境變數會覆寫程式碼中所設的同等值。

使用設定

您也可以使用 Configure方法設定適用於 Go 的 X-Ray 開發套件。 Configure 使用一個引數、一個Config物件，以及下列選用欄位。

DaemonAddr

此字串指定 X-Ray 協助程式接聽程式的主機和連接埠。如果未指定，X-Ray 會使用AWS_XRAY_DAEMON_ADDRESS環境變數的值。如果未設定該值，則會使用 "127.0.0.1:2000"。

ServiceVersion

此字串可指定服務的版本。如果未指定，X-Ray 會使用空字串 ("")。

SamplingStrategy

此 SamplingStrategy 物件可指定要追蹤哪些應用程式呼叫。如果未指定，X-Ray 會使用 LocalizedSamplingStrategy，採用 中定義的策略xray/resources/DefaultSamplingRules.json。

StreamingStrategy

此 StreamingStrategy 物件可指定當 RequiresStreaming 傳回 true 時是否要串流區段。如果未指定，X-Ray 會使用 DefaultStreamingStrategy，如果子區段數目大於 20，則會串流取樣的區段。

ExceptionFormattingStrategy

此 `ExceptionFormattingStrategy` 物件可指定您要如何處理各種例外狀況。如果未指定，X-Ray 會使用類型`DefaultExceptionFormattingStrategy`為 `XrayError`的 `error`、錯誤訊息和堆疊追蹤。

使用適用於 Go 的 X-Ray 開發套件檢測傳入 HTTP 請求

您可以使用 X-Ray 開發套件來追蹤您的應用程式在 Amazon EC2EC2 AWS Elastic Beanstalk或 Amazon ECS 中的 EC2 執行個體上提供的傳入 HTTP 請求。

使用 `xray.Handler` 檢測傳入的 HTTP 請求。適用於 Go 的 X-Ray 開發套件會在 `xray.Handler`類別中實作標準 Go 程式庫`http.Handler`界面，以攔截 Web 請求。`xray.Handler` 類別會使用請求的內容、剖析傳入標頭、視需要新增回應標頭，以將提供的 `http.Handler` 和 `xray.Capture` 包裝在一起並設定 HTTP 特定追蹤欄位。

當您使用此類別處理 HTTP 請求和回應時，適用於 Go 的 X-Ray 開發套件會為每個抽樣請求建立區段。此區段包括時間、方法，以及 HTTP 請求的處置方式。其他檢測會在此區段上建立子區段。

> **Note**
>
> 對於 AWS Lambda 函數，Lambda 會為每個抽樣請求建立區段。如需更多資訊，請參閱AWS Lambda 而且 AWS X-Ray。

下列範例會攔截連接埠 8000 上的請求，並傳回「Hello！」 作為回應。它會建立 myApp 區段，並檢測透過任何應用程式的呼叫。

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)
}
```

每個客群都有一個名稱，可在服務映射中識別您的應用程式。區段可以靜態命名，或者您可以設定 SDK，根據傳入請求中的主機標頭動態命名。動態命名可讓您根據請求中的網域名稱來分組追蹤，並在名稱不符合預期模式時套用預設名稱（例如，如果主機標頭是偽造的）。

> **轉送的請求**
>
> 如果負載平衡器或其他中介裝置轉送請求到您的應用程式，X-Ray 會從請求中的 `X-Forwarded-For`標頭取得用戶端 IP，而不是從 IP 封包中的來源 IP 取得用戶端 IP。為轉送請求記錄的用戶端 IP 可能是偽造的，因此不應受信任。

轉送請求時，軟體開發套件會在區段中設定額外的欄位來指出這一點。如果區段包含`x_forwarded_for`設為 的欄位`true`，用戶端 IP 會從 HTTP 請求中的 `X-Forwarded-For`標頭取得。

處理常式會使用 `http` 區塊為每個傳入的請求建立區段，其中包含以下資訊：

- HTTP 方法 – GET、POST、PUT、DELETE 等。
- 用戶端地址 – 傳送請求之用戶端的 IP 地址。
- 回應碼 – 已完成請求的 HTTP 回應碼。
- 時間 – 開始時間（收到請求的時間）和結束時間（傳送回應的時間）。
- 使用者代理程式 — 來自請求`user-agent`的 。
- 內容長度 — `content-length`來自回應的 。

設定區段命名策略

AWS X-Ray 使用服務名稱來識別您的應用程式，並將其與應用程式使用的其他應用程式、資料庫、外部 APIs AWS 和資源區分開來。當 X-Ray SDK 為傳入請求產生區段時，它會在區段的名稱[欄位中記錄應用程式的服務名稱](#)。

X-Ray SDK 可以在 HTTP 請求標頭中的主機名稱後面命名區段。不過，此標頭可以偽造，這可能會導致服務映射中出現非預期的節點。若要防止 SDK 因具有偽造主機標頭的請求而不正確命名區段，您必須指定傳入請求的預設名稱。

如果您的應用程式為多個網域提供請求，您可以將 SDK 設定為使用動態命名策略，以在區段名稱中反映這一點。動態命名策略可讓 SDK 將主機名稱用於符合預期模式的請求，並將預設名稱套用至不符合的請求。

例如，您可能有一個應用程式向三個子網域提供請求：www.example.com、 api.example.com和static.example.com。您可以使用動態命名策略搭配 模式*.example.com，以不同名稱識別每個子網域的區段，導致服務映射上有三個服務節點。如果您的應用程式收到主機名稱不符合模式的請求，您會在服務映射上看到具有您指定之備用名稱的第四個節點。

若要為所有請求區段使用相同的名稱，請如上一節所述，在建立處理常式時指定您應用程式的名稱。

> **Note**
>
> 您可以使用 AWS_XRAY_TRACING_NAME 環境變數來覆寫您在程式碼中定義的預設服務名稱。

動態命名策略可定義主機名稱應相符的模式，以及如果 HTTP 請求中的主機名稱不符合模式時要使用的預設名稱。若要動態命名區段，請使用 NewDynamicSegmentNamer 來設定要符合的預設名稱和模式。

Example main.go

如果請求中的主機名稱符合 *.example.com 模式，請使用該主機名稱。否則，請使用 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)
}
```

使用適用於 Go 的 X-Ray AWS 開發套件追蹤 SDK 呼叫

當您的應用程式呼叫 AWS 服務 來存放資料、寫入佇列或傳送通知時，適用於 Go 的 X-Ray 開發套件會在子區段中追蹤下游的呼叫。您在這些服務中存取的追蹤 AWS 服務 和資源 （ 例如 Amazon S3 儲存貯體或 Amazon SQS 佇列 ），會在 X-Ray 主控台的追蹤地圖上顯示為下游節點。

若要追蹤 AWS 開發套件用戶端，請使用 xray.AWS() 呼叫來包裝物件用戶端，如下範例所示。

Example main.go

```
var dynamo *dynamodb.DynamoDB
func main() {
```

```
  dynamo = dynamodb.New(session.Must(session.NewSession()))
  xray.AWS(dynamo.Client)
}
```

然後，當您使用 AWS 開發套件用戶端時，請使用 呼叫方法的 `withContext`版本，並從傳遞給處理常式的`contexthttp.Request`物件傳遞 。

Example main.go – AWS SDK 呼叫

```
func listTablesWithContext(ctx context.Context) {
  output := dynamo.ListTablesWithContext(ctx, &dynamodb.ListTablesInput{})
  doSomething(output)
}
```

對於所有 服務，您可以在 X-Ray 主控台中查看名為 的 API 名稱。對於服務子集，X-Ray SDK 會將資訊新增至區段，以在服務映射中提供更精細的 。

例如，當您使用經檢測的 DynamoDB 用戶端進行呼叫時，軟體開發套件會將資料表名稱新增至以資料表為目標的呼叫區段。在 主控台中，每個資料表都會在服務映射中顯示為個別節點，並具有一般 DynamoDB 節點，用於非以資料表為目標的呼叫。

Example 呼叫 DynamoDB 以儲存項目的子區段

```
{
  "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",
  }
}
```

您存取具名資源時，對以下服務的呼叫會在服務地圖中建立額外節點。未針對特定資源的呼叫，則會建立服務的一般節點。

- Amazon DynamoDB – 資料表名稱
- Amazon Simple Storage Service – 儲存貯體和金鑰名稱
- Amazon Simple Queue Service – 佇列名稱

使用 X-Ray SDK for Go 追蹤下游 HTTP Web 服務的呼叫

當您的應用程式呼叫微服務或公有 HTTP API 時，您可以使用 `xray.Client` 並以 Go 應用程式子區段的形式檢測這些呼叫，如下範例所示，其中 http-client 是 HTTP 用戶端。

用戶端會建立所提供 HTTP 用戶端的淺層副本，預設為 `http.DefaultClient`，其中三元式包裝為 `xray.RoundTripper`。

Example

<caption>main.go – HTTP 用戶端</caption>

```
myClient := xray.Client(http-client)
```

<caption>main.go – 使用 ctxhttp 程式庫追蹤下游 HTTP 呼叫</caption>

以下範例使用 使用 ctxhttp 程式庫來檢測傳出 HTTP 呼叫`xray.Client`。 `ctx` 可以從上游呼叫傳遞。這可確保使用現有的客群內容。例如，X-Ray 不允許在 Lambda 函數中建立新客群，因此應使用現有的 Lambda 客群內容。

```
resp, err := ctxhttp.Get(ctx, xray.Client(nil), url)
```

使用適用於 Go 的 X-Ray 開發套件追蹤 SQL 查詢

若要追蹤 PostgreSQL 或 MySQL 的 SQL 呼叫，請將 `sql.Open` 呼叫取代為 `xray.SQLContext`，如下範例所示。盡可能使用 URL，而非組態字串。

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
```

```
}
```

使用適用於 Go 的 X-Ray 開發套件產生自訂子區段

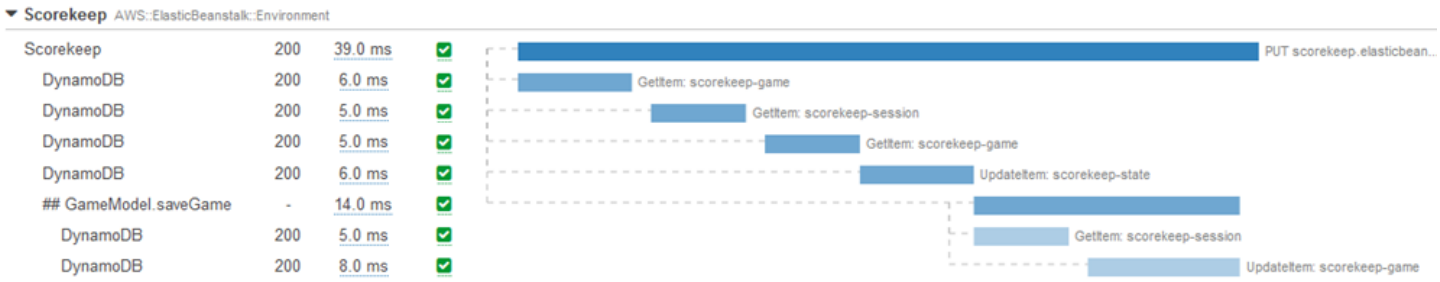
子區段會延伸追蹤的區段，其中包含為處理請求而完成之工作的詳細資訊。每次您與經檢測的用戶端進行呼叫時，X-Ray 開發套件都會記錄子區段中產生的資訊。您可以建立其他子區段來將其他子區段分組、測量程式碼區段的效能，或記錄註釋和中繼資料。

使用 `Capture` 方法來建立函數周圍的子區段。

Example main.go – 自訂子區段

```
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)
  })
```

以下螢幕擷取畫面顯示 `saveGame` 子區段在 `Scorekeep` 應用程式追蹤中可能出現的方式。



使用適用於 Go 的 X-Ray 開發套件將註釋和中繼資料新增至區段

您可以使用註釋和中繼資料記錄有關請求、環境或應用程式的其他資訊。您可以將註釋和中繼資料新增至 X-Ray 開發套件建立的區段，或新增至您建立的自訂子區段。

註釋是具有字串、數字或布林值的鍵值對。註釋會編製索引，以便與篩選條件表達式搭配使用。使用標記記錄您想要用來在主控台將追蹤分組的資料，或是在呼叫 `GetTraceSummaries` API 時使用標記。

中繼資料是索引鍵/值對，可以具有任何類型的值，包括物件和清單，但不會編製索引以用於篩選條件表達式。使用中繼資料記錄您希望儲存在追蹤中的其他資料，但不需要搭配搜尋使用。

除了註釋和中繼資料，您也可以在區段上記錄使用者 ID 字串。區段會將使用者 ID 記錄在單獨的欄位中，並建立索引以用於搜尋。

章節

- 使用適用於 Go 的 X-Ray 開發套件記錄註釋
- 使用適用於 Go 的 X-Ray 開發套件錄製中繼資料
- 使用適用於 Go IDs

使用適用於 Go 的 X-Ray 開發套件記錄註釋

針對您想要建立索引以用於搜尋的區段，請使用註釋來記錄這些區段上的資訊。

註釋要求

- 金鑰 – X-Ray 註釋的金鑰最多可有 500 個英數字元。您不能使用點或句號以外的空格或符號 (。)
- 值 – X-Ray 註釋的值最多可有 1，000 個 Unicode 字元。
- 註釋數量 – 每個追蹤最多可以使用 50 個註釋。

若要記錄註釋，請使用包含要與區段建立關聯之中繼資料的字串來呼叫 AddAnnotation。

```
xray.AddAnnotation(key string, value interface{})
```

軟體開發套件會將標註以鍵/值對記錄在區段文件中的 annotations 物件內。若使用相同索引鍵呼叫 AddAnnotation 兩次，則會覆寫之前在相同區段上記錄的值。

若要尋找具有特定值註釋的追蹤，請在篩選條件表達式中使用 annotation[*key*]關鍵字。

使用適用於 Go 的 X-Ray 開發套件錄製中繼資料

針對您不想要建立索引以用於搜尋的區段，請使用中繼資料來記錄這些區段上的資訊。

若要記錄中繼資料，請使用包含要與區段建立關聯之中繼資料的字串來呼叫 AddMetadata。

```
xray.AddMetadata(key string, value interface{})
```

使用適用於 Go IDs

記錄請求區段上的使用者 ID 以識別傳送請求的使用者。

記錄使用者 ID

1. 從 AWSXRay 取得目前區段的參考。

```
import (
  "context"
  "github.com/aws/aws-xray-sdk-go/xray"
)

mySegment := xray.GetSegment(context)
```

2. 使用傳送請求之使用者的字串 ID 呼叫 `setUser`。

```
mySegment.User = "U12345"
```

若要尋找使用者 ID 的追蹤，請在篩選條件表達式中使用 `user`關鍵字。

使用 Java

有兩種方式可以檢測您的 Java 應用程式，以將追蹤傳送至 X-Ray：

- AWS Distro for OpenTelemetry Java – 透過 AWS Distro for OpenTelemetry Collector， AWS 提供一組開放原始碼程式庫，用於將相關指標和追蹤傳送至多個 AWS 監控解決方案，包括 Amazon CloudWatch AWS X-Ray和 Amazon OpenSearch Service。
- AWS X-Ray 適用於 Java 的 SDK – 一組程式庫，用於透過 X-Ray 協助程式產生追蹤並將其傳送至 X-Ray。

如需詳細資訊，請參閱選擇 AWS Distro for OpenTelemetry 和 X-Ray SDKs。

AWS Distro for OpenTelemetry Java

使用 AWS Distro for OpenTelemetry (ADOT) Java，您可以檢測您的應用程式一次，並將相關指標和追蹤傳送至多個 AWS 監控解決方案 AWS X-Ray，包括 Amazon CloudWatch 和 Amazon OpenSearch Service。將 X-Ray 與 ADOT 搭配使用需要兩個元件：啟用 OpenTelemetry SDK 以與 X-Ray 搭配使用，以及啟用 AWS Distro for OpenTelemetry Collector 以與 X-Ray 搭配使用。ADOT Java 包含自動檢測支援，讓您的應用程式無需變更程式碼即可傳送追蹤。

若要開始使用，請參閱 AWS Distro for OpenTelemetry Java 文件。

如需搭配 AWS AWS X-Ray 和其他 使用 Distro for OpenTelemetry 的詳細資訊 AWS 服務，請參閱 AWS Distro for OpenTelemetry 或 AWS Distro for OpenTelemetry 文件。

如需語言支援和用量的詳細資訊，請參閱 AWS GitHub 上的可觀測性。

AWS X-Ray 適用於 Java 的 SDK

適用於 Java 的 X-Ray 開發套件是一組適用於 Java Web 應用程式的程式庫，提供類別和方法來產生追蹤資料並傳送至 X-Ray 協助程式。追蹤資料包含應用程式所服務傳入 HTTP 請求的相關資訊，以及應用程式使用 AWS SDK、HTTP 用戶端或 SQL 資料庫連接器對下游服務進行的呼叫。您也可以手動建立區段，並將除錯資訊新增至註釋和中繼資料中。

適用於 Java 的 X-Ray 開發套件是開放原始碼專案。您可以追蹤專案，並在 GitHub 上提交問題與提取請求：github.com/aws/aws-xray-sdk-java

首先，將 AWSXRayServletFilter 新增為 Servlet 篩選條件以追蹤傳入的請求。servlet 篩選條件會建立區段。當區段開啟時，您可以使用軟體開發套件用戶端的方法，將資訊新增到區段，並建立子區段以追蹤下游呼叫。軟體開發套件也會在區段為開啟時自動記錄應用程式擲回的例外狀況。

從 1.3 版開始，您可以使用 Spring 的切面導向程式設計 (AOP) 來檢測應用程式。這表示您可以在應用程式執行時檢測應用程式 AWS，而無需將任何程式碼新增至應用程式的執行時間。

接下來，使用適用於 Java 的 X-Ray 開發套件，透過在建置組態中包含 SDK Instrumentor 子模組 適用於 Java 的 AWS SDK 來檢測用戶端。每當您使用經檢測的用戶端呼叫下游 AWS 服務 或資源時，開發套件會在子區段中記錄有關呼叫的資訊。 AWS 服務 而您在服務中存取的資源會在追蹤地圖上顯示為下游節點，以協助您識別錯誤並調節個別連線的問題。

如果您不想檢測所有下游呼叫 AWS 服務，您可以離開 Instrumentor 子模組，然後選擇要檢測的用戶端。將 新增至 TracingHandler AWS SDK 服務用戶端，以檢測個別用戶端。

其他適用於 Java 的 X-Ray 開發套件子模組提供對 HTTP Web APIs和 SQL 資料庫進行下游呼叫的檢測。您可以在 Apache HTTP 子模組中使用適用於 Java 的 X-Ray 開發套件 HTTPClient和 HTTPClientBuilder 版本來檢測 Apache HTTP 用戶端。若要檢測 SQL 查詢，請將軟體開發套件的攔截程式新增至資料來源。

開始使用 SDK 之後，請設定記錄器和 servlet 篩選條件來自訂其行為。您可以新增外掛程式，以記錄執行應用程式所需的運算資源相關資料、定義抽樣規則以自訂抽樣行為，並設定日誌層級以在應用程式日誌中查看更多或更少的軟體開發套件資訊。

使用註釋與中繼資料，記錄應用程式所做的請求和工作等其他資訊。註釋是簡單的鍵/值對，系統會為其建立索引以用於篩選條件表達式，因此您可以搜尋包含特定資料的追蹤。中繼資料項目的限制性較低，可以記錄整個物件和陣列，任何可以序列化為 JSON 的項目。

> **標註與中繼資料**
>
> 註釋和中繼資料是您使用 X-Ray SDK 新增至區段的任意文字。註釋會編製索引，以便與篩選條件表達式搭配使用。中繼資料不會編製索引，但可以使用 X-Ray 主控台或 API 在原始區段中檢視。您授予 X-Ray 讀取存取權的任何人都可以檢視此資料。

當程式碼中有許多經過檢測的用戶端時，單一請求區段可能包含大量子區段，每個使用經檢測用戶端進行的呼叫都有一個子區段。您可以將用戶端呼叫包裝在自訂子區段中，以組織和群組子區段。您可以為整個函數或任何部分的程式碼建立自訂子區段，並記錄子區段上的中繼資料和註釋，而不必寫入父區段上的所有項目。

子模組

您可以從 Maven 下載適用於 Java 的 X-Ray 開發套件。適用於 Java 的 X-Ray 開發套件會依使用案例分割為子模組，其中包含版本管理的物料清單：

- `aws-xray-recorder-sdk-core`（必要）– 用於建立客群和傳輸客群的基本功能。包含 `AWSXRayServletFilter` 以檢測傳入的請求。
- `aws-xray-recorder-sdk-aws-sdk` – 將追蹤 適用於 Java 的 AWS SDK 用戶端新增為請求處理常式，以檢測對用戶端 AWS 服務 發出的呼叫。
- `aws-xray-recorder-sdk-aws-sdk-v2` – 將追蹤用戶端新增為請求攔截器，以檢測對 適用於 Java 的 AWS SDK 2.2 和更新版本的用戶端 AWS 服務 發出的呼叫。
- `aws-xray-recorder-sdk-aws-sdk-instrumentor` – 使用 `aws-xray-recorder-sdk-aws-sdk`， 會自動檢測所有 適用於 Java 的 AWS SDK 用戶端。
- `aws-xray-recorder-sdk-aws-sdk-v2-instrumentor` – 使用 `aws-xray-recorder-sdk-aws-sdk-v2`時， 會自動檢測所有 適用於 Java 的 AWS SDK 2.2 和更新版本的用戶端。
- `aws-xray-recorder-sdk-apache-http` – 檢測使用 Apache HTTP 用戶端進行的傳出 HTTP 呼叫。
- `aws-xray-recorder-sdk-spring` – 為 Spring AOP Framework 應用程式提供攔截器。
- `aws-xray-recorder-sdk-sql-postgres` – 檢測使用 JDBC 對 PostgreSQL 資料庫發出的傳出呼叫。
- `aws-xray-recorder-sdk-sql-mysql` – 檢測使用 JDBC 對 MySQL 資料庫發出的傳出呼叫。
- `aws-xray-recorder-sdk-bom` – 提供物料清單，可用來指定用於所有子模組的版本。
- `aws-xray-recorder-sdk-metrics` – 從收集的 X-Ray 區段發佈未取樣的 Amazon CloudWatch 指標。

如果您使用 Maven 或 Gradle 建置應用程式，請將適用於 Java 的 X-Ray 開發套件新增至您的建置組態。

如需 SDK 類別和方法的參考文件，請參閱AWS X-Ray 適用於 Java 的 SDK API 參考。

要求

適用於 Java 的 X-Ray 開發套件需要 Java 8 或更新版本、Servlet API 3、 AWS SDK 和 Jackson。

軟體開發套件在編譯和執行時間需仰賴下列程式庫：

- AWS 適用於 Java 的 SDK 1.11.398 版或更新版本
- Servlet API 3.1.0

軟體開發套件的 pom.xml 檔案中有宣告這些相依性。如果您使用 Maven 或 Gradle 來建置，則會自動包含這些相依性。

如果您使用的程式庫包含在適用於 Java 的 X-Ray 開發套件中，則必須使用隨附的版本。例如，如果您已在執行時間相依於 Jackson 並為了該相依性在部署中包含 JAR 檔案，則必須移除這些 JAR 檔案，因為軟體開發套件 JAR 包含自己的 Jackson 程式庫版本。

相依性管理

適用於 Java 的 X-Ray 開發套件可從 Maven 取得：

- 群組 – com.amazonaws
- 成品 – aws-xray-recorder-sdk-bom
- 版本：2.11.0

如果您使用 Maven 來建置應用程式，請在 pom.xml 檔案中，將軟體開發套件新增為依存項目。

Example pom.xml - 依存項目

```
<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>
```

若是 Gradle，請在 `build.gradle` 檔案中，將軟體開發套件新增為編譯時間依存項目。

Example build.gradle - 相依性

```
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 部署應用程式，則可以使用 Maven 或 Gradle 在每次部署時建置執行個體，而不是建置和上傳包含所有相依性的大型封存。如需使用 Gradle 的範例，請參閱範例應用程式。

AWS X-Ray 適用於 Java 的自動檢測代理程式

適用於 Java 的 AWS X-Ray 自動檢測代理程式是一種追蹤解決方案，只需最少的開發工作即可檢測您的 Java Web 應用程式。代理程式可追蹤 servlet 型應用程式，以及代理程式使用支援的架構和程式庫提出的所有下游請求。這包括下游 Apache HTTP 請求、 AWS SDK 請求，以及使用 JDBC 驅動程式進行的 SQL 查詢。代理程式會在執行緒之間傳播 X-Ray 內容，包括所有作用中區段和子區段。X-Ray SDK 的所有組態和多樣性仍然可以搭配 Java 代理程式使用。已選擇適合的預設值，以確保代理程式以最少的工作量運作。

X-Ray 代理程式解決方案最適合以 servlet 為基礎的請求回應 Java Web 應用程式伺服器。如果您的應用程式使用非同步架構，或是未妥善建模為請求回應服務，建議您改為考慮使用 SDK 進行手動檢測。

X-Ray 代理程式是使用分散式系統理解工具組或 DiSCo 建置。DiSCo 是用於建置 Java 代理程式的開放原始碼架構，可用於分散式系統。雖然您不需要了解 DiSCo 是否使用 X-Ray 代理程式，但您可以前往其 GitHub 首頁進一步了解專案。X-Ray 代理程式也是完全開放原始碼。若要檢視原始程式碼、做出貢獻或引發有關代理程式的問題，請造訪 GitHub 上的儲存庫。

範例應用程式

eb-java-scorekeep 範例應用程式已調整為使用 X-Ray 代理程式進行檢測。此分支不包含 servlet 篩選條件或記錄器組態，因為這些函數是由代理程式完成。若要在本機或使用 AWS 資源執行應用程式，請遵循範例應用程式讀我檔案的步驟。使用範例應用程式產生 X-Ray 追蹤的指示，請參閱範例應用程式的教學課程。

開始使用

若要在您自己的應用程式中開始使用 X-Ray 自動檢測 Java 代理程式，請遵循下列步驟。

1. 在您的環境中執行 X-Ray 協助程式。如需詳細資訊，請參閱AWS X-Ray 協助程式。
2. 下載代理程式的最新分佈。解壓縮封存，並記下其在檔案系統中的位置。其內容應如下所示。

```
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. 修改應用程式的 JVM 引數，以包含下列項目，以啟用代理程式。如果適用，請確定`-javaagent`引數放在`-jar`引數前面。修改 JVM 引數的程序會根據您用來啟動 Java 伺服器的工具與架構而有所不同。如需特定指引，請參閱伺服器架構的文件。

```
-javaagent:/<path-to-disco>/disco-java-agent.jar=pluginPath=/<path-to-disco>/disco-plugins
```

4. 若要指定應用程式的名稱在 X-Ray 主控台上顯示的方式，請設定`AWS_XRAY_TRACING_NAME`環境變數或`com.amazonaws.xray.strategy.tracingName`系統屬性。如果未提供名稱，則會使用預設名稱。
5. 重新啟動您的伺服器或容器。現在會追蹤傳入請求及其下游呼叫。如果您沒有看到預期的結果，請參閱 the section called “故障診斷”。

組態

X-Ray 代理程式是由外部使用者提供的 JSON 檔案所設定。根據預設，此檔案位於名為 的使用者 classpath 根目錄（例如，在其`resources`目錄中）`xray-agent.json`。您可以將`com.amazonaws.xray.configFile`系統屬性設定為組態檔案的絕對檔案系統路徑，以設定組態檔案的自訂位置。

接下來會顯示範例組態檔案。

```
{
    "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
}
```

組態規格

下表說明每個屬性的有效值。屬性名稱區分大小寫，但其金鑰不區分大小寫。對於可由環境變數和系統屬性覆寫的屬性，優先順序一律是環境變數，然後是系統屬性，然後是組態檔案。如需可覆寫屬性的相關資訊，請參閱 環境變數。所有欄位都是選擇性的。

屬性名稱	Type	有效值	描述	環境變數	系統屬性	預設
serviceName	字串	任何字串	在 X-Ray 主控台中顯示的受檢測服務名稱。	AWS_XRAY_TRACING_NAME	com.amazonaws.xray.strategy.tracingName	XRayInstrumentedService
contextMissingStrategy	字串	LOG_ERROR、IGNORE_ERROR	當代理程式嘗試使用 X-Ray 區段內容，但不存在時，代理程式所採取的動作。	AWS_XRAY_CONTEXT_MISSING	com.amazonaws.xray.strategy.contextMissingStrategy	LOG_ERROR
daemonAddress	字串	格式化 IP 地址和連接埠，或 TCP 和 UDP 地址清單	代理程式用來與 X-Ray 協助程式通訊的地址。	AWS_XRAY_DAEMON_ADDRESS	com.amazonaws.xray.emitter.daemonAddress	127.0.0.1：2000
tracingEnabled	Boolean	True、False	啟用 X-Ray 代理程式的檢測。	AWS_XRAY_TRACING_ENABLED	com.amazonaws.xray.tracingEnabled	TRUE
samplingStrategy	字串	CENTRAL、LOCAL、NONE、ALL	代理程式使用的取樣策略。ALL 會擷取所有請	N/A	N/A	中樞

屬性名稱	Type	有效值	描述	環境變數	系統屬性	預設
			求，NONE 不會擷取任何請求。請參閱取樣規則。			
traceIdInjectionPrefix	字串	任何字串	在 日誌中包含注入追蹤 IDs前提供的字首。	N/A	N/A	無（空字串）
samplingRulesManifest	字串	絕對檔案路徑	自訂抽樣規則檔案的路徑，做為本機抽樣策略的抽樣規則來源，或中央策略的備用規則。	N/A	N/A	DefaultSamplingRules.json
awsServiceHandlerManifest	字串	絕對檔案路徑	自訂參數允許清單的路徑，該清單會從 AWS SDK 用戶端擷取其他資訊。	N/A	N/A	DefaultOperationParameterWhitelist.json

屬性名稱	Type	有效值	描述	環境變數	系統屬性	預設
awsSdkVersion	Integer	1、2	您正在使用的AWS 適用於 Java 的開發套件版本。如果 也未設定`awsServieHandlerManifest`，則忽略。	N/A	N/A	2
maxStackTraceLength	Integer	非負整數	要在追蹤中記錄的堆疊追蹤的最大行數。	N/A	N/A	50
streamingThreshold	Integer	非負整數	至少關閉此多個子區段後，它們會串流到out-of-band的協助程式，以避免區塊太大。	N/A	N/A	100

屬性名稱	Type	有效值	描述	環境變數	系統屬性	預設
traceIdInjection	Boolean	True、False	如果也新增了記錄組態所述的相依性和組態，則啟用 X-Ray 追蹤 ID ???注入日誌。否則，不會執行任何動作。	N/A	N/A	TRUE
pluginsEnabled	Boolean	True、False	啟用可記錄您正在操作 AWS 環境中繼資料的外掛程式。請參閱 外掛程式。	N/A	N/A	TRUE
collectSqlQueries	Boolean	True、False	盡力在 SQL 子區段中記錄 SQL 查詢字串。	N/A	N/A	FALSE

屬性名稱	Type	有效值	描述	環境變數	系統屬性	預設
contextPropagation	Boolean	True、False	如果為 true，在執行緒之間自動傳播 X-Ray 內容。否則，會使用 Thread Local 來存放內容，並需要手動跨執行緒傳播。	N/A	N/A	TRUE

記錄組態

X-Ray 代理程式的日誌層級的設定方式與適用於 Java 的 X-Ray 開發套件相同。日誌 如需使用適用於 Java 的 X-Ray 開發套件設定記錄的詳細資訊，請參閱 。

手動檢測

如果您想要在代理程式的自動檢測之外執行手動檢測，請將 X-Ray 開發套件新增為專案的相依性。請注意，追蹤傳入請求中提到的 SDK 自訂 servlet 篩選條件與 X-Ray 代理程式不相容。

> **Note**
>
> 您必須使用最新版本的 X-Ray 開發套件來執行手動檢測，同時還使用 代理程式。

如果您在 Maven 專案中工作，請將下列相依性新增至您的 `pom.xml`檔案。

```
<dependencies>
  <dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-xray-recorder-sdk-core</artifactId>
    <version>2.11.0</version>
```

```
  </dependency>
  </dependencies>
```

如果您在 Gradle 專案中工作，請將下列相依性新增至您的 `build.gradle`檔案。

```
implementation 'com.amazonaws:aws-xray-recorder-sdk-core:2.11.0'
```

除了註釋、中繼資料和使用者 IDs 之外，您還可以在使用代理程式時新增自訂子區段，就像使用一般 SDK 一樣。代理程式會自動跨執行緒傳播內容，因此在使用多執行緒應用程式時，不需要傳播內容的解決方法。

故障診斷

由於代理程式提供全自動檢測，當您遇到問題時，可能很難識別問題的根本原因。如果 X-Ray 代理程式無法如預期般運作，請檢閱下列問題和解決方案。X-Ray 代理程式和 SDK 使用 Jakarta Commons Logging (JCL)。若要查看記錄輸出，請確定將 JCL 連接至記錄後端的橋接器位於 classpath 上，如下列範例所示： `log4j-jcl`或 `jcl-over-slf4j`。

問題：我已在我的應用程式上啟用 Java 代理程式，但在 X-Ray 主控台上看不到任何內容

X-Ray 協助程式是否在同一部機器上執行？

如果沒有，請參閱 X-Ray 協助程式文件進行設定。

在您的應用程式日誌中，您是否看到像是「初始化 X-Ray 代理程式記錄器」的訊息？

如果您已將代理程式正確新增至您的應用程式，則此訊息會在應用程式啟動時，於開始接收請求之前記錄在 INFO 層級。如果此訊息不存在，則 Java 代理程式不會與 Java 程序一起執行。請確定您已正確遵循所有設定步驟，且沒有錯別字。

在您的應用程式日誌中，您是否看到數個錯誤訊息，指出「隱藏 AWS X-Ray 內容遺漏例外狀況」？

這些錯誤是因為代理程式嘗試檢測下游請求，例如 AWS SDK 請求或 SQL 查詢，但代理程式無法自動建立客群。如果您看到其中許多錯誤，代理程式可能不是最適合您的使用案例的工具，建議您改為考慮使用 X-Ray SDK 進行手動檢測。或者，您可以啟用 X-Ray SDK 偵錯日誌，以查看發生內容遺失例外狀況的堆疊追蹤。您可以使用自訂區段包裝程式碼的這些部分，這應該會解決這些錯誤。如需使用自訂區段包裝下游請求的範例，請參閱範例程式碼檢測啟動程式碼。

問題：我預期的某些區段不會出現在 X-Ray 主控台上

您的應用程式是否使用多執行緒？

如果您預期建立的某些客群未出現在主控台中，則應用程式中的背景執行緒可能是原因。如果您的應用程式使用背景執行緒「觸發並忘記」來執行任務，例如使用 AWS SDK 對 Lambda 函數進行一次性呼叫，或定期輪詢某些 HTTP 端點，這可能會在代理程式跨執行緒傳播內容時混淆代理程式。若要驗證這是您的問題，請啟用 X-Ray SDK 偵錯日誌，並檢查是否有訊息，例如：未發出名為 <NAME > 的區段，因為其父系正在進行的子區段。若要解決此問題，您可以嘗試在伺服器傳回之前加入背景執行緒，以確保記錄其中完成的所有工作。或者，您可以將代理程式的`contextPropagation`組態設定為`false`，以停用背景執行緒中的內容傳播。如果您這樣做，則必須使用自訂區段手動檢測這些執行緒，或忽略其產生的缺少內容例外狀況。

您是否已設定抽樣規則？

如果 X-Ray 主控台上似乎出現隨機或非預期的區段，或您預期在主控台上的區段未出現，您可能會遇到抽樣問題。X-Ray 代理程式會使用 X-Ray 主控台的規則，將集中式取樣套用至其建立的所有區段。預設規則是每秒 1 個區段，加上之後 5% 的區段，會進行取樣。這表示可能無法取樣使用代理程式快速建立的區段。若要解決此問題，您應該在 X-Ray 主控台上建立自訂取樣規則，以適當取樣所需的區段。如需詳細資訊，請參閱取樣。

設定適用於 Java 的 X-Ray 開發套件

適用於 Java 的 X-Ray 開發套件包含名為 的類別`AWSXRay`，可提供全域記錄器。這是可以用來檢測程式碼的 `TracingHandler`。您可以設定全域記錄器來自訂為傳入 HTTP 呼叫建立區段的`AWSXRayServletFilter`。

章節

- 服務外掛程式
- 抽樣規則
- 日誌
- 區段接聽程式
- 環境變數
- 系統屬性

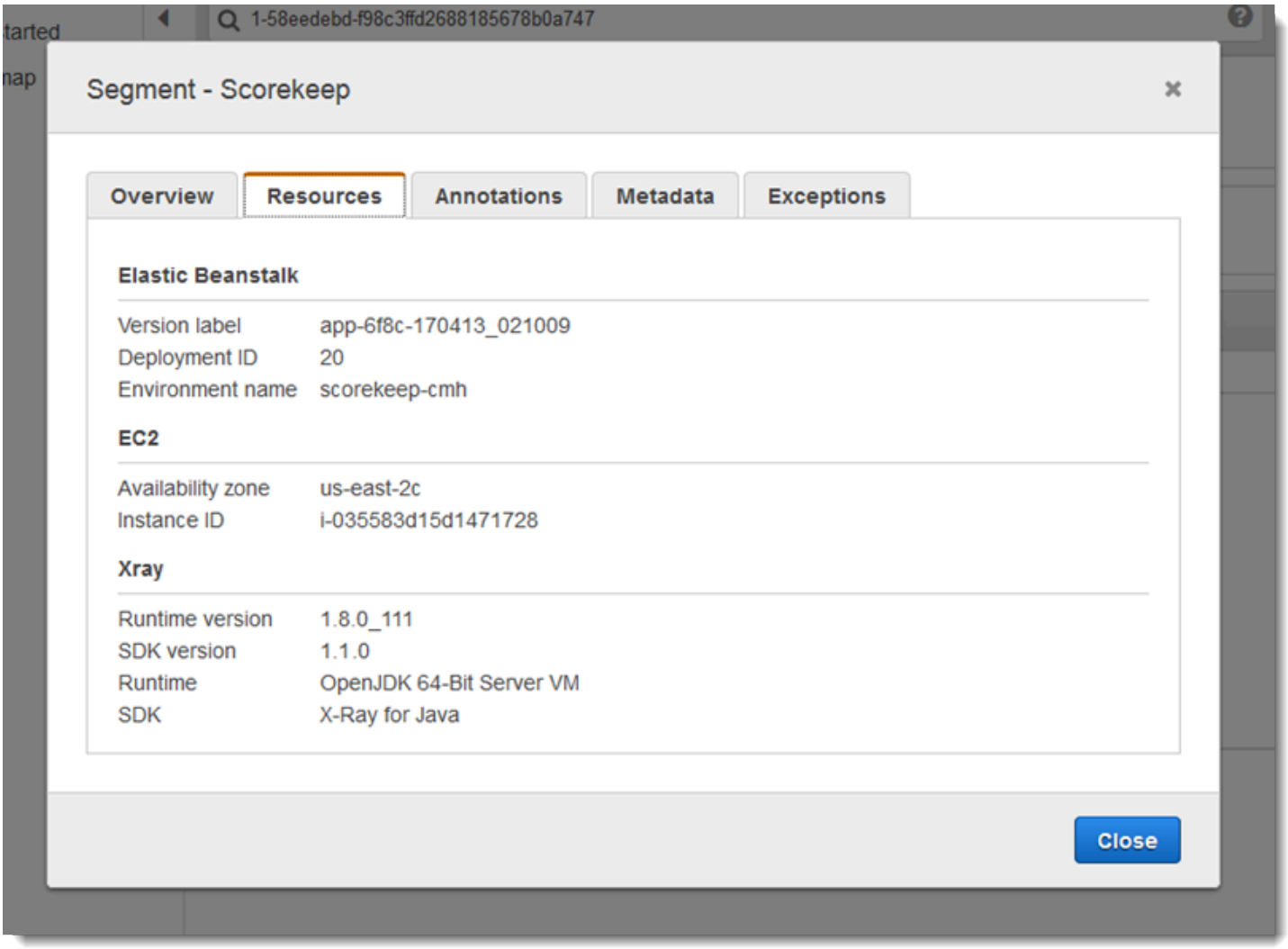
服務外掛程式

使用 `plugins` 記錄託管您應用程式之服務的相關資訊。

外掛程式

- Amazon EC2 – `EC2Plugin` 新增執行個體 ID、可用區域和 CloudWatch Logs 群組。

- Elastic Beanstalk – `ElasticBeanstalkPlugin` 新增環境名稱、版本標籤和部署 ID。
- Amazon ECS – `ECSPlugin` 新增容器 ID。
- Amazon EKS – `EKSPlugin` 新增容器 ID、叢集名稱、Pod ID 和 CloudWatch Logs 群組。



若要使用外掛程式，請在 `AWSXRayRecorderBuilder` 上呼叫 `withPlugin`。

Example src/main/java/scorekeep/WebConfig.java - 記錄器

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

軟體開發套件也會使用外掛程式設定來設定區段上的 `origin` 欄位。這表示執行您應用程式 AWS 的資源類型。當您使用多個外掛程式時，開發套件會使用下列解析順序來判斷原始伺服器：ElasticBeanstalk > EKS > ECS > EC2。

抽樣規則

SDK 會使用您在 X-Ray 主控台中定義的抽樣規則來決定要記錄哪些請求。預設規則每秒追蹤第一個請求，以及所有傳送追蹤至 X-Ray 服務的任何其他請求的 5%。在 X-Ray 主控台中建立其他規則，以自訂為每個應用程式記錄的資料量。

軟體開發套件會依定義自訂規則的順序進行套用。如果請求符合多個自訂規則，軟體開發套件只會套用第一個規則。

> **Note**
>
> 如果開發套件無法達到 X-Ray 以取得取樣規則，它會每秒還原為第一個請求的預設本機規則，以及每個主機任何額外請求的 5%。如果主機沒有呼叫取樣 APIs許可，或無法連線至 X-Ray 協助程式，而該常駐程式可做為 SDK 進行 API 呼叫的 TCP 代理，則可能會發生這種情況。

您也可以設定 SDK 以從 JSON 文件載入取樣規則。開發套件可以使用本機規則作為 X-Ray 取樣不可用的備份，或僅使用本機規則。

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
  }
}
```

此範例會定義一個自訂規則和預設規則。自訂規則會套用 5% 的取樣率，沒有追蹤 下路徑的最低請求數/api/move/。預設規則會追蹤每秒的第一個請求和 10% 的額外請求。

在本機定義規則的缺點是，固定目標是由記錄器的每個執行個體獨立套用，而不是由 X-Ray 服務管理。當您部署更多主機時，固定速率會乘以，使得難以控制記錄的資料量。

在 上 AWS Lambda，您無法修改取樣率。如果您的函數是由 受檢測的服務呼叫，則 Lambda 會記錄產生該服務取樣請求的呼叫。如果啟用主動追蹤，且不存在追蹤標頭，Lambda 會做出抽樣決策。

若要在 Spring 中提供備份規則，請在組態類別中使用 CentralizedSamplingStrategy 設定全域記錄器：

Example src/main/java/myapp/WebConfig.java - 記錄器組態

```
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());
}
```

若是 Tomcat，請新增接聽程式以擴展 `ServletContextListener`，並在部署描述項中註冊接聽程式。

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>
```

若僅要使用本機規則，請將 CentralizedSamplingStrategy 取代為 LocalizedSamplingStrategy。

```
builder.withSamplingStrategy(new LocalizedSamplingStrategy(ruleFile));
```

日誌

根據預設，開發套件會將 ERROR層級訊息輸出到您的應用程式日誌。您可以在 SDK 上啟用偵錯層級記錄，將更詳細的日誌輸出至應用程式日誌檔案。有效的日誌層級為 DEBUG、INFO、ERROR、 WARN和 FATAL。FATAL日誌層級會靜音所有日誌訊息，因為 SDK 不會在嚴重層級進行日誌。

Example application.properties

使用 logging.level.com.amazonaws.xray 屬性設定記錄日誌層級。

```
logging.level.com.amazonaws.xray = DEBUG
```

當您手動產生子區段時，可使用除錯日誌來識別問題，例如未結束的子區段。

將追蹤 ID 插入日誌

若要將您日誌陳述式公開給目前的追蹤 ID，您可以將此 ID 插入到映射的診斷內容 (MDC)。使用 SegmentListener 界面，會在區段生命週期事件期間從 X-Ray 記錄器呼叫方法。當區段或子區段開始時，合格的追蹤 ID 會透過金鑰 AWS-XRAY-TRACE-ID 注入至 MDC。當該區段結束時，該索引鍵即會從 MDC 中移除。這會公開正在使用的記錄程式庫追蹤 ID。當子區段結束時，其父 ID 會注入 MDC 中。

Example 完整的合格追蹤 ID

完整的合格 ID 會表示為 TraceID@EntityID

```
1-5df42873-011e96598b447dfca814c156@541b3365be3dafc3
```

此功能適用於使用適用於 Java 的 AWS X-Ray 開發套件檢測的 Java 應用程式，並支援下列記錄組態：

- SLF4J 前端 API 與 Logback 後端
- SLF4J 前端 API 與 Log4J2 後端
- Log4J2 前端 API 與 Log4J2 後端

請參閱下列標籤，以了解每個前端和每個後端的需求。

SLF4J Frontend

1. 將以下 Maven 相依性新增到您的專案。

```
<dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-xray-recorder-sdk-slf4j</artifactId>
    <version>2.11.0</version>
</dependency>
```

2. 建置 AWSXRayRecorder 時包含 withSegmentListener 方法。這會新增 SegmentListener 類別，將新的追蹤 ID 自動插入 SLF4J MDC。

   SegmentListener 會採用選擇性字串做為參數來設定日誌陳述式的字首。您可以透過下列方式設定字首：

   - 無 – 使用預設AWS-XRAY-TRACE-ID字首。
   - 空白 – 使用空字串 （例如 ""）。
   - 自訂 – 使用字串中定義的自訂字首。

   Example **AWSXRayRecorderBuilder** 陳述式

```
AWSXRayRecorderBuilder builder = AWSXRayRecorderBuilder
        .standard().withSegmentListener(new SLF4JSegmentListener("CUSTOM-
PREFIX"));
```

Log4J2 front end

1. 將以下 Maven 相依性新增到您的專案。

```
<dependency>
    <groupId>com.amazonaws</groupId>
```

```
    <artifactId>aws-xray-recorder-sdk-log4j</artifactId>
    <version>2.11.0</version>
</dependency>
```

2. 建置 AWSXRayRecorder 時包含 withSegmentListener 方法。這會新增 SegmentListener 類別，將新的完整合格追蹤 ID 自動注入 SLF4J MDC。

   SegmentListener 會採用選擇性字串做為參數來設定日誌陳述式的字首。您可以透過下列方式設定字首：

   - 無 – 使用預設AWS-XRAY-TRACE-ID字首。
   - 空白 – 使用空字串 （ 例如 "") 並移除字首。
   - 自訂 – 使用字串中定義的自訂字首。

   Example **AWSXRayRecorderBuilder** 陳述式

```
AWSXRayRecorderBuilder builder = AWSXRayRecorderBuilder
        .standard().withSegmentListener(new Log4JSegmentListener("CUSTOM-
PREFIX"));
```

Logback backend

若要將追蹤 ID 插入到日誌事件中，您必須修改記錄器的 PatternLayout，這會格式化每個記錄陳述式。

1. 尋找 patternLayout 的設定位置。您可透過編寫程式的方式，或透過 XML 組態檔案執行此作業。若要深入了解，請參閱 Logback 組態。
2. 將 %X{AWS-XRAY-TRACE-ID} 插入到 patternLayout 的任何位置，可將追蹤 ID 插入未來的記錄陳述式。%X{} 表示您正在使用 MDC 提供的索引鍵擷取值。若要深入了解 Logback 中的 PatternLayout，請參閱 PatternLayout。

Log4J2 backend

1. 尋找 patternLayout 的設定位置。您可透過編寫程式的方式，或透過以 XML、JSON、YAML 或屬性格式編寫的組態檔案執行此作業。

   若要深入了解如何透過組態檔案設定 Log4J2，請參閱組態。

若要深入了解如何透過編寫程式的方式設定 Log4J2，請參閱透過編寫程式方式的組態。

2. 將 %X{AWS-XRAY-TRACE-ID} 插入到 PatternLayout 的任何位置，可將追蹤 ID 插入未來的記錄陳述式。%X{} 表示您正在使用 MDC 提供的索引鍵擷取值。若要深入了解 Log4J2 中的 PatternLayout，請參閱模式配置。

插入追蹤 ID 範例

以下示範包含追蹤 ID 的修改後 PatternLayout 字串。追蹤 ID 會列印在執行緒名稱 (%t) 之後、日誌層級 (%-5p) 之前。

Example 插入 ID 的 **PatternLayout**

```
%d{HH:mm:ss.SSS} [%t] %X{AWS-XRAY-TRACE-ID} %-5p %m%n
```

AWS X-Ray 會自動列印日誌陳述式中的金鑰和追蹤 ID，以便於剖析。以下顯示使用修改後 PatternLayout 的日誌說明。

Example 插入 ID 的日誌說明

```
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
```

記錄訊息本身以 %m 模式包裝，在呼叫記錄器時設定。

區段接聽程式

區段接聽程式是攔截生命週期事件的界面，例如 產生的區段開始和結束AWSXRayRecorder。區段接聽程式事件函數的實作可能是在透過 onBeginSubsegment 建立時，將相同的註釋新增至所有子區段、使用 afterEndSegment 將每個區段傳送到精靈後記錄訊息，或者透過 beforeEndSubsegment記錄由 SQL 攔截器傳送的查詢，以驗證子區段是否代表 SQL 查詢，如果是的話，則會新增額外的中繼資料。

若要查看SegmentListener函數的完整清單，請造訪AWS X-Ray 適用於 Java 的 記錄器開發套件 API 的文件。

下列範例顯示如何在建立 onBeginSubsegment 時將一致的註釋加入所有子區段，以及透過 afterEndSegment 在每個區段結尾列印記錄訊息。

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());
    }
}
```

在建立 AWSXRayRecorder 時參考此自訂區段接聽程式。

Example AWSXRayRecorderBuilder statement

```
AWSXRayRecorderBuilder builder = AWSXRayRecorderBuilder
        .standard().withSegmentListener(new MySegmentListener());
```

環境變數

您可以使用環境變數來設定適用於 Java 的 X-Ray 開發套件。軟體開發套件支援以下變數。

- AWS_XRAY_CONTEXT_MISSING – 設定為 RUNTIME_ERROR 以在未開啟區段時，檢測程式碼嘗試記錄資料時擲回例外狀況。

  有效值

  - RUNTIME_ERROR – 捨棄執行時間例外狀況。
  - LOG_ERROR – 記錄錯誤並繼續 （預設）。
  - IGNORE_ERROR – 忽略錯誤並繼續。

當您嘗試在未開啟請求時執行的啟動程式碼中，或在產生新執行緒的程式碼中，使用經檢測的用戶端時，可能會發生與缺少區段或子區段相關的錯誤。

- AWS_XRAY_DAEMON_ADDRESS – 設定 X-Ray 協助程式接聽程式的主機和連接埠。依預設，軟體開發套件會使用 127.0.0.1:2000 進行追蹤資料 (UDP) 和取樣 (TCP)。如果您已設定協助程式在不同的連接埠上接聽，或正在不同的主機上執行，請使用此變數。

  格式

  - 相同連接埠 – *address*:*port*
  - 不同的連接埠 – tcp:*address*:*port* udp:*address*:*port*
- AWS_LOG_GROUP – 將日誌群組的名稱設定為與您的應用程式相關聯的日誌群組。如果您的日誌群組使用與您應用程式相同的 AWS 帳戶和區域，X-Ray 將使用此指定的日誌群組自動搜尋應用程式的區段資料。如需日誌群組的詳細資訊，請參閱使用日誌群組和串流。
- AWS_XRAY_TRACING_NAME – 設定 SDK 用於區段的服務名稱。覆寫您在 servlet 篩選條件的區段命名策略中設定的服務名稱。

環境變數會覆寫程式碼中所設的同等系統屬性和值。

系統屬性

您可以將系統屬性做為環境變數的 JVM 專用替代方案。開發套件支援以下屬性：

- com.amazonaws.xray.strategy.tracingName – 相當於 AWS_XRAY_TRACING_NAME。
- com.amazonaws.xray.emitters.daemonAddress – 相當於 AWS_XRAY_DAEMON_ADDRESS。
- com.amazonaws.xray.strategy.contextMissingStrategy – 相當於 AWS_XRAY_CONTEXT_MISSING。

如果同時設定了系統屬性和同等環境變數，則會使用環境變數的值。兩種方法都會覆寫程式碼中所設的值。

使用適用於 Java 的 X-Ray 開發套件追蹤傳入請求

您可以使用 X-Ray 開發套件來追蹤您的應用程式在 Amazon EC2EC2 AWS Elastic Beanstalk或 Amazon ECS 中的 EC2 執行個體上提供的傳入 HTTP 請求。

使用 `Filter` 檢測傳入的 HTTP 請求。當您將 X-Ray servlet 篩選條件新增至應用程式時，適用於 Java 的 X-Ray 開發套件會為每個抽樣請求建立區段。此區段包括時間、方法，以及 HTTP 請求的處置方式。其他檢測會在此區段上建立子區段。

> **Note**
>
> 對於 AWS Lambda 函數，Lambda 會為每個抽樣請求建立區段。如需更多資訊，請參閱AWS Lambda 而且 AWS X-Ray。

每個客群都有一個名稱，可在服務映射中識別您的應用程式。區段可以靜態命名，或者您可以設定 SDK，根據傳入請求中的主機標頭動態命名。動態命名可讓您根據請求中的網域名稱來分組追蹤，並在名稱不符合預期模式時套用預設名稱（例如，如果主機標頭是偽造的）。

> **轉送的請求**
>
> 如果負載平衡器或其他中介裝置轉送請求到您的應用程式，X-Ray 會從請求中的 `X-Forwarded-For`標頭取得用戶端 IP，而不是從 IP 封包中的來源 IP 取得用戶端 IP。為轉送請求記錄的用戶端 IP 可能是偽造的，因此不應受信任。

轉送請求時，軟體開發套件會在區段中設定額外的欄位來指出這一點。如果區段包含`x_forwarded_for`設為 的欄位`true`，用戶端 IP 會從 HTTP 請求中的 `X-Forwarded-For`標頭取得。

訊息處理常式會使用 `http` 區塊為每個傳入的請求建立區段，其中包含以下資訊：

- HTTP 方法 – GET、POST、PUT、DELETE 等。
- 用戶端地址 – 傳送請求之用戶端的 IP 地址。
- 回應碼 – 已完成請求的 HTTP 回應碼。
- 時間 – 開始時間（收到請求的時間）和結束時間（傳送回應的時間）。
- 使用者代理程式 — 來自請求`user-agent`的 。
- 內容長度 — `content-length`來自回應的 。

章節

- 將追蹤篩選條件新增至應用程式 (Tomcat)

- 將追蹤篩選條件新增至應用程式 (Spring)
- 設定區段命名策略

將追蹤篩選條件新增至應用程式 (Tomcat)

若是 Tomcat，請將 `<filter>` 新增至您專案的 `web.xml` 檔案。使用 `fixedName` 參數，指定要套用至針對傳入請求建立之區段的服務名稱。

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>
```

將追蹤篩選條件新增至應用程式 (Spring)

若是 Spring，請將 `Filter` 新增至您的 `WebConfig` 類別。以字串形式將區段名稱傳遞給 `AWSXRayServletFilter` 建構函數。

Example src/main/java/myapp/WebConfig.java - Spring

```
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");
  }
}
```

設定區段命名策略

AWS X-Ray 使用服務名稱來識別您的應用程式，並將其與應用程式使用的其他應用程式、資料庫、外部 APIs AWS 和資源區分開來。當 X-Ray SDK 為傳入請求產生區段時，它會在區段的名稱欄位中記錄應用程式的服務名稱。

X-Ray SDK 可以在 HTTP 請求標頭中的主機名稱後面命名區段。不過，此標頭可以偽造，這可能會導致服務映射中出現非預期的節點。若要防止 SDK 因具有偽造主機標頭的請求而不正確命名區段，您必須指定傳入請求的預設名稱。

如果您的應用程式為多個網域提供請求，您可以將 SDK 設定為使用動態命名策略，以在區段名稱中反映這一點。動態命名策略可讓 SDK 將主機名稱用於符合預期模式的請求，並將預設名稱套用至不符合的請求。

例如，您可能有一個應用程式向三個子網域提供請求：`www.example.com`、 `api.example.com`和 `static.example.com`。您可以使用動態命名策略搭配 模式`*.example.com`，以不同名稱識別每個子網域的區段，導致服務映射上有三個服務節點。如果您的應用程式收到主機名稱不符合模式的請求，您會在服務映射上看到具有您指定之備用名稱的第四個節點。

若要為所有請求區段使用相同的名稱，請如上一節所述，在初始化 servlet 篩選條件時指定您應用程式的名稱。這與透過呼叫`SegmentNamingStrategy.fixed()`並傳遞給建構函數來建立固定 SegmentNamingStrategy `AWSXRayServletFilter` 有相同的效果。

> **Note**
>
> 您可以使用 `AWS_XRAY_TRACING_NAME` 環境變數來覆寫您在程式碼中定義的預設服務名稱。

動態命名策略可定義主機名稱應相符的模式，以及如果 HTTP 請求中的主機名稱不符合模式時要使用的預設名稱。若要在 Tomcat 中動態命名區段，請分別使用 `dynamicNamingRecognizedHosts` 和 `dynamicNamingFallbackName` 來定義模式和預設名稱。

Example WEB-INF/web.xml - 使用動態命名的 Servlet 篩選條件

```
<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>
```

對於 Spring，呼叫 來建立動態 SegmentNamingStrategySegmentNamingStrategy.dynamic()，並將其傳遞給AWSXRayServletFilter建構函數。

Example src/main/java/myapp/WebConfig.java - 使用動態命名的 Servlet 篩選條件

```
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"));
  }
}
```

使用適用於 Java 的 X-Ray AWS 開發套件追蹤 SDK 呼叫

當您的應用程式呼叫 AWS 服務 來存放資料、寫入佇列或傳送通知時，適用於 Java 的 X-Ray 開發套件會在子區段中追蹤下游的呼叫。您在這些服務中存取的追蹤 AWS 服務 和資源 （ 例如 Amazon S3 儲存貯體或 Amazon SQS 佇列），會在 X-Ray 主控台的追蹤映射上顯示為下游節點。

當您在建置中包含 aws-sdk和 aws-sdk-instrumentor子模組時，適用於 Java 的 X-Ray 開發套件會自動檢測所有 AWS 開發套件用戶端。如果您未包含 Instrumentor 子模組，您可以選擇檢測某些特定用戶端，而排除其他用戶端。

若要檢測個別用戶端，請從您的建置中移除 aws-sdk-instrumentor子模組，並使用 服務的用戶端建置器，在 AWS SDK 用戶端TracingHandler上新增 XRayClient做為 。

例如，若要檢測 AmazonDynamoDB 用戶端，請將追蹤處理常式傳遞至 AmazonDynamoDBClientBuilder。

Example MyModel.java - DynamoDB 用戶端

```
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();
...
```

對於所有 服務，您可以在 X-Ray 主控台中查看名為 的 API 名稱。對於服務子集，X-Ray SDK 會將資訊新增至區段，以在服務映射中提供更精細的 。

例如，當您使用經檢測的 DynamoDB 用戶端進行呼叫時，軟體開發套件會將資料表名稱新增至以資料表為目標的呼叫區段。在 主控台中，每個資料表都會在服務映射中顯示為個別節點，並具有一般 DynamoDB 節點，用於非以資料表為目標的呼叫。

Example 呼叫 DynamoDB 以儲存項目的子區段

```
{
  "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",
  }
}
```

您存取具名資源時，對以下服務的呼叫會在服務地圖中建立額外節點。未針對特定資源的呼叫，則會建立服務的一般節點。

- Amazon DynamoDB – 資料表名稱
- Amazon Simple Storage Service – 儲存貯體和金鑰名稱
- Amazon Simple Queue Service – 佇列名稱

若要 AWS 服務 使用 適用於 Java 的 AWS SDK 2.2 和更新版本檢測對 的下游呼叫，您可以從建置組態中省略 `aws-xray-recorder-sdk-aws-sdk-v2-instrumentor`模組。這時改成包含 `aws-xray-recorder-sdk-aws-sdk-v2 module`，然後使用 `TracingInterceptor` 為其進行設定，檢測個別的用戶端。

Example 適用於 Java 的 AWS SDK 2.2 及更新版本 - 追蹤攔截器

```
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();
//...
```

使用適用於 Java 的 X-Ray 開發套件追蹤對下游 HTTP Web 服務的呼叫

當您的應用程式呼叫微服務或公有 HTTP APIs，您可以使用適用於 Java 的 X-Ray 開發套件 版本`HttpClient`來檢測這些呼叫，並將 API 做為下游服務新增至服務圖表。

適用於 Java 的 X-Ray 開發套件包含 `DefaultHttpClient`和 `HttpClientBuilder`類別，可用於取代 Apache HttpComponents 對等項目，以檢測傳出的 HTTP 呼叫。

- `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`

這些程式庫位於 [aws-xray-recorder-sdk-apache-http] 子模組中。

您可以使用相當於檢測所有用戶端的 X-Ray 取代現有的匯入陳述式，或在初始化用戶端以檢測特定用戶端時使用完整名稱。

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();
    }
  }
```

當您檢測對下游 Web api 的呼叫時，適用於 Java 的 X-Ray 開發套件會記錄子區段，其中包含 HTTP 請求和回應的相關資訊。X-Ray 使用 子區段來產生遠端 API 的推斷區段。

Example 下游 HTTP 呼叫的子區段

```
{
  "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 呼叫的推斷區段

```
{
  "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
}
```

使用適用於 Java 的 X-Ray 開發套件追蹤 SQL 查詢

SQL 攔截器

將適用於 Java JDBC 的 X-Ray 開發套件攔截器新增至資料來源組態，以檢測 SQL 資料庫查詢。

- PostgreSQL – `com.amazonaws.xray.sql.postgres.TracingInterceptor`
- MySQL – `com.amazonaws.xray.sql.mysql.TracingInterceptor`

這些攔截器分別位於 aws-xray-recorder-sql-postgres 和 aws-xray-recorder-sql-mysql 子模組中。他們會實作 `org.apache.tomcat.jdbc.pool.JdbcInterceptor`，並且與 Tomcat 連線集區相容。

> **Note**
>
> 基於安全性目的，SQL 攔截器不會在子區段內記錄 SQL 查詢本身。

針對 Spring，請在屬性檔案中新增攔截器，然後使用 Spring Boot 的 `DataSourceBuilder` 建置資料來源。

Example **`src/main/java/resources/application.properties`** - PostgreSQL JDBC 攔截器

```
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`** - 資料來源

```
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();
  }
...
}
```

對於 Tomcat，請在 JDBC 資料來源`setJdbcInterceptors`上呼叫 ，並參考適用於 Java 的 X-Ray 開發套件類別。

Example **`src/main/myapp/model.java`** - 資料來源

```
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 資料來源程式庫包含在適用於 Java 的 X-Ray 開發套件中，但您可以將其宣告為提供的相依性，以記錄您使用它。

Example **pom.xml** - JDBC 資料來源

```
<dependency>
  <groupId>org.apache.tomcat</groupId>
  <artifactId>tomcat-jdbc</artifactId>
  <version>8.0.36</version>
  <scope>provided</scope>
</dependency>
```

原生 SQL 追蹤裝飾項目

- 將 aws-xray-recorder-sdk-sql新增至您的相依性。
- 裝飾資料庫資料來源、連線或陳述式。

```
dataSource = TracingDataSource.decorate(dataSource)
connection = TracingConnection.decorate(connection)
statement = TracingStatement.decorateStatement(statement)
preparedStatement = TracingStatement.decoratePreparedStatement(preparedStatement,
 sql)
callableStatement = TracingStatement.decorateCallableStatement(callableStatement,
 sql)
```

使用適用於 Java 的 X-Ray 開發套件產生自訂子區段

子區段會延伸追蹤的區段，其中包含為處理請求而完成之工作的詳細資訊。每次您與經檢測的用戶端進行呼叫時，X-Ray 開發套件都會記錄子區段中產生的資訊。您可以建立其他子區段來將其他子區段分組、測量程式碼區段的效能，或記錄註釋和中繼資料。

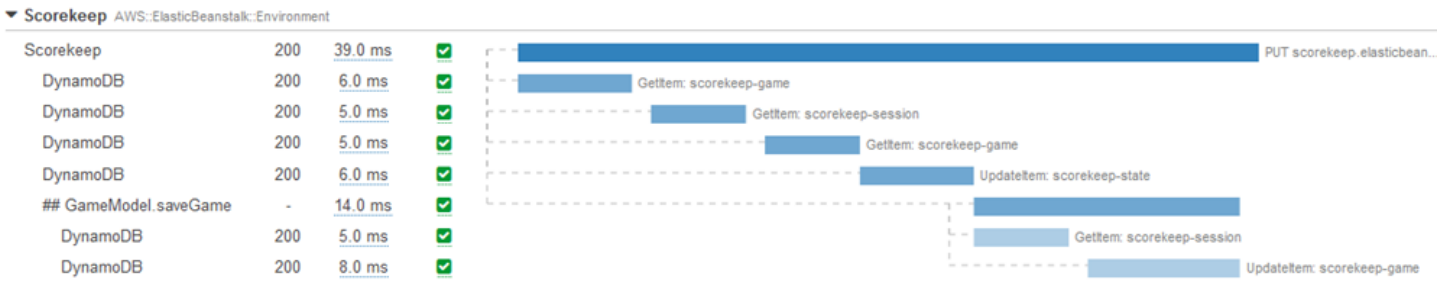
若要管理子區段，請使用 beginSubsegment 和 endSubsegment 方法。

Example GameModel.java - 自訂子區段

```
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();
    }
  }
```

在此範例中，子區段中的程式碼會使用工作階段模型上的 方法，從 DynamoDB 載入遊戲的工作階段，並使用 適用於 Java 的 AWS SDK的 DynamoDB 映射器來儲存遊戲。在 主控台的追蹤檢視中，將此程式碼包裝在子區段中會呼叫Save Game子區段的 DynamoDB 子系。



如果子區段中的程式碼擲回已檢查的例外狀況，請將其包裝在 `try` 區塊中，並在 `finally` 區塊中呼叫 `AWSXRay.endSubsegment()`，以確保子區段一律關閉。如果子區段未關閉，則無法完成父區段，也不會傳送至 X-Ray。

對於未擲回核取例外狀況的程式碼，您可以將程式碼以 Lambda 函數`AWSXRay.CreateSubsegment`的形式傳遞至 。

Example 子區段 Lambda 函數

```
import com.amazonaws.xray.AWSXRay;

AWSXRay.createSubsegment("getMovies", (subsegment) -> {
    // function code
});
```

當您在區段或其他子區段中建立子區段時，適用於 Java 的 X-Ray 開發套件會為其產生 ID，並記錄開始時間和結束時間。

Example 使用中繼資料的子區段

```
"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"
    }
  },
```

對於非同步和多執行緒程式設計，您必須手動將子區段傳遞至 endSubsegment()方法，以確保其正確關閉，因為 X-Ray 內容可能會在非同步執行期間修改。如果非同步子區段在其父區段關閉後關閉，此方法會自動將整個區段串流到 X-Ray 協助程式。

Example 非同步子區段

```
@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();
}
```

使用適用於 Java 的 X-Ray 開發套件將註釋和中繼資料新增至區段

您可以使用註釋和中繼資料記錄有關請求、環境或應用程式的其他資訊。您可以將註釋和中繼資料新增至 X-Ray 開發套件建立的區段，或新增至您建立的自訂子區段。

註釋是具有字串、數字或布林值的鍵值對。註釋會編製索引，以便與篩選條件表達式搭配使用。使用標記記錄您想要用來在主控台將追蹤分組的資料，或是在呼叫 `GetTraceSummaries` API 時使用標記。

中繼資料是索引鍵/值對，可以具有任何類型的值，包括物件和清單，但不會編製索引以用於篩選條件表達式。使用中繼資料記錄您希望儲存在追蹤中的其他資料，但不需要搭配搜尋使用。

除了註釋和中繼資料，您也可以在區段上記錄使用者 ID 字串。區段會將使用者 ID 記錄在單獨的欄位中，並建立索引以用於搜尋。

章節

- 使用適用於 Java 的 X-Ray 開發套件記錄註釋
- 使用適用於 Java 的 X-Ray 開發套件記錄中繼資料
- 使用適用於 Java IDs

使用適用於 Java 的 X-Ray 開發套件記錄註釋

針對您想要建立索引以用於搜尋的區段或子區段，請使用標註來記錄這些區段上的資訊。

註釋要求

- 金鑰 – X-Ray 註釋的金鑰最多可有 500 個英數字元。您不能使用點或句號以外的空格或符號 (。)
- 值 – X-Ray 註釋的值最多可有 1，000 個 Unicode 字元。
- 註釋數量 – 每個追蹤最多可以使用 50 個註釋。

記錄標註

1. 從 AWSXRay 取得目前區段或子區段的參考。

```
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. 使用字串索引鍵、布林值、數字或字串值，呼叫 putAnnotation。

```
document.putAnnotation("mykey", "my value");
```

下列範例顯示如何使用包含點的putAnnotation字串索引鍵和布林值、數字或字串值來呼叫 。

```
document.putAnnotation("testkey.test", "my value");
```

軟體開發套件會將標註以鍵/值對記錄在區段文件中的 annotations 物件內。若使用相同鍵呼叫 putAnnotation 兩次，則會覆寫之前在相同區段或子區段上記錄的值。

若要尋找具有特定值註釋的追蹤，請在篩選條件表達式中使用 annotation[*key*]關鍵字。

Example **src/main/java/scorekeep/GameModel.java** – 註釋和中繼資料

```
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();
    }
  }
```

使用適用於 Java 的 X-Ray 開發套件記錄中繼資料

針對您不想要建立索引以用於搜尋的區段，請使用中繼資料來記錄這些區段或子區段上的資訊。中繼資料值可以是字串、數字、布林值，或可序列化為 JSON 物件或陣列的任何物件。

記錄中繼資料

1. 從 AWSXRay 取得目前區段或子區段的參考。

   ```
   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. 使用字串命名空間、字串索引鍵、布林值、數字、字串或物件值，呼叫 putMetadata。

   ```
   document.putMetadata("my namespace", "my key", "my value");
   ```

   或

   只使用鍵和值呼叫 putMetadata。

   ```
   document.putMetadata("my key", "my value");
   ```

若您沒有指定命名空間，軟體開發套件會使用 default。若使用相同鍵呼叫 putMetadata 兩次，則會覆寫之前在相同區段或子區段上記錄的值。

Example **src/main/java/scorekeep/GameModel.java** – 註釋和中繼資料

```
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();
    }
  }
```

使用適用於 Java IDs

記錄請求區段上的使用者 ID 以識別傳送請求的使用者。

記錄使用者 ID

1. 從 AWSXRay 取得目前區段的參考。

   ```
   import com.amazonaws.xray.AWSXRay;
   import com.amazonaws.xray.entities.Segment;
   ...
   Segment document = AWSXRay.getCurrentSegment();
   ```

2. 使用傳送請求之使用者的字串 ID 呼叫 `setUser`。

```
document.setUser("U12345");
```

您可以在控制器中呼叫 setUser，以在應用程式開始處理請求時馬上記錄使用者 ID。如果您只要使用區段來設定使用者 ID，可以將呼叫鏈結為單行。

Example src/main/java/scorekeep/MoveController.java – 使用者 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);
  }
```

若要尋找使用者 ID 的追蹤，請在篩選條件表達式中使用 user關鍵字。

AWS X-Ray 適用於 Java 的 X-Ray 開發套件的 指標

本主題說明 AWS X-Ray 命名空間、指標和維度。您可以使用適用於 Java 的 X-Ray 開發套件，從收集的 X-Ray 區段發佈未取樣的 Amazon CloudWatch 指標。這些指標衍生自區段的開始和結束時間，以及錯誤、故障和節流狀態標記。您可以使用這些追蹤指標，公開子區段內的重試和相依性問題。

CloudWatch 是指標儲存庫。指標是 CloudWatch 中的基本概念，代表一組按時間排序的資料點。您（或 AWS 服務) 將指標資料點發佈至 CloudWatch，並以一組循序的時間序列資料擷取有關這些資料點的統計資料。

指標是由名稱、命名空間和一個或多個維度做唯一的定義。每個資料點都有時間戳記和可選的測量單位。當您請求統計資料時，傳回的資料流是藉由命名空間、指標名稱和維度做識別。

如需有關 CloudWatch 的詳細資訊，請參閱《Amazon CloudWatch 使用者指南》。

X-Ray CloudWatch 指標

ServiceMetrics/SDK 命名空間包含下列指標。

指標	統計資訊可用	描述	個單位
Latency	平均、最小、最大、計數	開始與結束時間之間的差異。平均、最小和最大皆描述操作延遲。計數描述呼叫計數。	毫秒
ErrorRate	平均數、總和	失敗原因為 4xx Client Error 狀態碼的請求速率，導致錯誤。	百分比
FaultRate	平均數、總和	失敗原因為 5xx Server Error 狀態碼的追蹤速率，導致故障。	百分比
ThrottleRate	平均數、總和	傳回 429 狀態碼的節流追蹤速率。這是 ErrorRate 指標的一部分。	百分比
OkRate	平均數、總和	產生 OK 狀態碼的追蹤請求速率。	百分比

X-Ray CloudWatch 維度

使用下表中的維度來精簡針對 X-Ray 檢測Java應用程式傳回的指標。

維度	描述
ServiceType	如不清楚，即為服務的類型，例如，AWS::EC2::Instance 或 NONE。
ServiceName	服務的正式名稱。

啟用 X-Ray CloudWatch 指標

使用下列程序，在受檢測的Java應用程式中啟用追蹤指標。

設定追蹤指標

1. 新增`aws-xray-recorder-sdk-metrics`套件做為Apache Maven相依性。如需詳細資訊，請參閱適用於 Java 子模組的 X-Ray 開發套件。
2. 啟用新的 `MetricsSegmentListener()` 做為全域記錄器組建的一部分。

   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 代理程式以使用 Amazon Elastic Compute Cloud (Amazon EC2)、Amazon Elastic Container Service (Amazon ECS) 或 Amazon Elastic Kubernetes Service (Amazon EKS) 收集指標：

   - 若要設定 Amazon EC2，請參閱安裝 CloudWatch 代理程式。

- 若要設定 Amazon ECS，請參閱使用 Container Insights 監控 Amazon ECS 容器。
- 若要設定 Amazon EKS，請參閱使用 Amazon CloudWatch 可觀測性 Amazon CloudWatch 代理程式。

4. 設定 SDK 以與 CloudWatch 代理程式通訊。根據預設，軟體開發套件會與地址 上的 CloudWatch 代理程式通訊`127.0.0.1`。您可以將環境變數或 Java 屬性設為 `address:port`，以設定替代位址。

   Example 環境變數

   ```
   AWS_XRAY_METRICS_DAEMON_ADDRESS=address:port
   ```

   Example Java 屬性

   ```
   com.amazonaws.xray.metrics.daemonAddress=address:port
   ```

驗證組態

1. 登入 AWS Management Console ，並在 https：//https://console.aws.amazon.com/cloudwatch/ 開啟 CloudWatch 主控台。
2. 開啟 Metrics (指標) 標籤，以觀察指標的湧入。
3. （選用）在 CloudWatch 主控台的日誌索引標籤上，開啟`ServiceMetricsSDK`日誌群組。尋找符合主機指標的日誌資料流，並確認日誌訊息。

在多執行緒應用程式之間傳遞區段內容

當您在應用程式中建立新執行緒時，`AWSXRayRecorder` 不會維持目前區段或子區段實體的參考。若您在新執行緒中使用受檢測的用戶端，軟體開發套件會嘗試寫入不存在的區段，且會導致 SegmentNotFoundException。

若要避免在開發期間拋出異常，您可以使用 ContextMissingStrategy 設定記錄器，告知其改而記錄錯誤。您可以使用 SetContextMissingStrategy 在程式碼中設定策略，或是使用環境變數或系統屬性來設定相等選項。

其中一種處理錯誤的方法是透過在啟動執行緒時呼叫 beginSegment 來使用新區段，以及在關閉時呼叫 endSegment。若您要檢測並非針對回應 HTTP 請求而執行的程式碼，這種方法可以正常運作，就像在應用程式啟動時執行的程式碼。

若您使用多個執行緒來處理傳入請求，您可以將目前區段或子區段傳遞至新執行緒，並將它提供給全域記錄器。這可確保在記錄該請求的其餘資訊時，於新執行緒中記錄的資訊會與相同區段建立關聯。一旦區段可在新執行緒中使用，您就可以使用 `segment.run(() -> { ... })`方法執行可存取該區段內容的任何可執行項目。

如需範例，請參閱在工作者執行緒中使用受檢測用戶端。

搭配非同步程式設計使用 X-Ray

適用於 Java 的 X-Ray 開發套件可用於具有 SegmentContextExecutors 的非同步 Java 程式。SegmentContextExecutor 實作執行器界面，這表示它可以傳遞到 CompletableFuture 的所有非同步操作。這可確保任何非同步操作都會在其內容中使用正確的區段執行。

Example App.java 範例：將 SegmentContextExecutor 傳遞至 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());
```

搭配 Spring 和適用於 Java 的 X-Ray 開發套件的 AOP

本主題說明如何使用 X-Ray SDK 和 Spring Framework 來檢測您的應用程式，而無需變更其核心邏輯。這表示現在有一種非侵入性的方式來檢測從遠端執行的應用程式 AWS。

啟用 Spring 中的 AOP

1. 設定 Spring
2. 將追蹤篩選條件新增至您的應用程式
3. 對您的程式碼做註釋或實作界面
4. 啟用應用程式中的 X-Ray

設定 Spring

您可以使用 Maven 或 Gradle 將 Spring 設定為使用 AOP 檢測您的應用程式。

如果您使用 Maven 來建置應用程式，請在 `pom.xml` 檔案中，新增以下相依性。

```
<dependency>
     <groupId>com.amazonaws</groupId>
     <artifactId>aws-xray-recorder-sdk-spring</artifactId>
     <version>2.11.0</version>
</dependency>
```

若是 Gradle，請在 `build.gradle` 檔案中，新增以下依存性。

```
compile 'com.amazonaws:aws-xray-recorder-sdk-spring:2.11.0'
```

設定 Spring Boot

除了上一節所述的 Spring 相依性之外，如果您使用的是 Spring Boot，如果尚未在 classpath 上，請新增下列相依性。

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'
```

將追蹤篩選條件新增至您的應用程式

將 `Filter`新增至您的 `WebConfig` 類別。以字串形式將區段名稱傳遞給 `AWSXRayServletFilter` 建構函數。如需追蹤篩選條件和檢測傳入請求的詳細資訊，請參閱 使用適用於 Java 的 X-Ray 開發套件追蹤傳入請求。

Example src/main/java/myapp/WebConfig.java - Spring

```
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");
  }
}
```

雅加達支援

Spring 6 使用 Jakarta 而非 Javax 做為其 Enterprise Edition。為了支援這個新的命名空間，X-Ray 已建立一組平行的類別，這些類別位於自己的雅加達命名空間中。

對於篩選條件類別，請將 取代javax為 jakarta。設定區段命名策略時，請在命名策略類別名稱jakarta之前新增 ，如下列範例所示：

```
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"));
    }
}
```

對您的程式碼做註釋或實作界面

您的類別必須以 @XRayEnabled註釋標註，或實作 XRayTraced 界面。這會通知 AOP 系統，針對 X-Ray 檢測包裝受影響的類別函數。

在應用程式中啟用 X-Ray

若要在應用程式中啟用 X-Ray 追蹤，您的程式碼必須透過覆寫下列方法BaseAbstractXRayInterceptor來擴展抽象類別。

- generateMetadata- 此函數允許自訂連接至目前函數追蹤的中繼資料。根據預設，執行函數的類別名稱會記錄到中繼資料中。如果您需要其他資訊，可以新增更多資料。
- xrayEnabledClasses- 此函數是空的，應保持為空。可做為 pointcut 的主機，指示攔截程式有哪些包裝方法。指定哪些類別要使用 @XRayEnabled 做註釋以便追蹤，藉此定義 pointcut。以下 pointcut 陳述式會通知攔截程式包裝所有含 @XRayEnabled 註釋的控制器 Bean。

```
@Pointcut(“@within(com.amazonaws.xray.spring.aop.XRayEnabled) && bean(*Controller)”)
```

如果您的專案使用 Spring Data JPA，請考慮從 延伸，AbstractXRayInterceptor而非 BaseAbstractXRayInterceptor。

範例

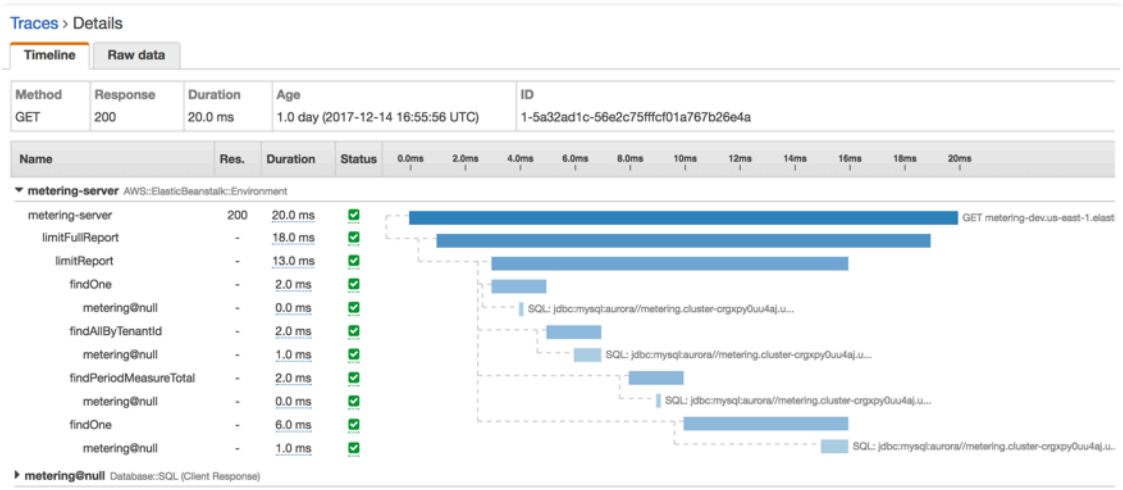
下列程式碼延伸抽象類別 BaseAbstractXRayInterceptor。

```
@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() {}

}
```

下列程式碼為將由 X-Ray 檢測的類別。

```
@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());
        }
    }
}
```

如果您已正確設定應用程式，則應該會看到應用程式的完整呼叫堆疊 (從控制器到服務呼叫)，如以下主控台的螢幕擷取畫面所示。



使用 Node.js

有兩種方式可以檢測 Node.js 應用程式，將追蹤傳送至 X-Ray：

- AWS Distro for OpenTelemetry JavaScript – 透過 AWS Distro for OpenTelemetry Collector， AWS 提供一組開放原始碼程式庫，用於將相關指標和追蹤傳送至多個 AWS 監控解決方案，包括 Amazon CloudWatch AWS X-Ray和 Amazon OpenSearch Service。
- AWS X-Ray 適用於 Node.js 的 SDK – 一組程式庫，用於透過 X-Ray 協助程式產生和傳送追蹤至 X-Ray。

如需詳細資訊，請參閱選擇 AWS Distro for OpenTelemetry 和 X-Ray SDKs。

AWS Distro for OpenTelemetry JavaScript

使用 AWS Distro for OpenTelemetry (ADOT) JavaScript，您可以檢測您的應用程式一次，並將相關指標和追蹤傳送至多個 AWS 監控解決方案，包括 Amazon CloudWatch AWS X-Ray和 Amazon OpenSearch Service。搭配 AWS Distro for OpenTelemetry 使用 X-Ray 需要兩個元件：啟用 OpenTelemetry SDK 以搭配 X-Ray 使用，以及啟用 AWS Distro for OpenTelemetry Collector 以搭配 X-Ray 使用。

若要開始使用，請參閱 AWS Distro for OpenTelemetry JavaScript 文件。

> **Note**
>
> 所有伺服器端 Node.js 應用程式都支援 ADOT JavaScript。ADOT JavaScript 無法從瀏覽器用戶端將資料匯出至 X-Ray。

如需搭配 AWS AWS X-Ray 和其他 使用 Distro for OpenTelemetry 的詳細資訊 AWS 服務，請參閱 AWS Distro for OpenTelemetry 或 AWS Distro for OpenTelemetry 文件。

如需語言支援和用量的詳細資訊，請參閱 AWS GitHub 上的可觀測性。

AWS 適用於 Node.js 的 X-Ray 開發套件

適用於 Node.js 的 X-Ray 開發套件是 Express Web 應用程式和 Node.js Lambda 函數的程式庫，提供產生追蹤資料並將其傳送至 X-Ray 協助程式的類別和方法。追蹤資料包含應用程式所提供服務之傳入 HTTP 請求的相關資訊，以及應用程式使用 AWS SDK 或 HTTP 用戶端對下游服務發出的呼叫。

> **Note**
>
> 適用於 Node.js 的 X-Ray 開發套件是 Node.js 14.x 版及更新版本支援的開放原始碼專案。您可以關注專案，並在 GitHub 上提交問題與提取 (pull) 請求：github.com/aws/aws-xray-sdk-node

若您使用 Express，請在您的應用程式伺服器上從將軟體開發套件新增為中介軟體開始，來追蹤傳入請求。中介軟體會為每個追蹤的請求建立區段，並在傳送回應時完成區段。當區段開啟時，您可以使用軟體開發套件用戶端的方法，將資訊新增到區段，並建立子區段以追蹤下游呼叫。軟體開發套件也會在區段為開啟時自動記錄應用程式擲回的例外狀況。

對於由經檢測的應用程式或服務呼叫的 Lambda 函數，Lambda 會自動讀取追蹤標頭並追蹤取樣的請求。對於其他 函數，您可以設定 Lambda 來取樣和追蹤傳入的請求。無論哪種情況，Lambda 都會建立區段，並將其提供給 X-Ray 開發套件。

> **Note**
>
> 在 Lambda 上，X-Ray 開發套件是選用的。如果您未在函數中使用它，您的服務映射仍會包含 Lambda 服務的節點，以及每個 Lambda 函數的一個節點。透過新增 SDK，您可以檢測函數程式碼，將子區段新增至 Lambda 記錄的函數區段。如需更多資訊，請參閱AWS Lambda 而且 AWS X-Ray。

接著，使用適用於 Node.js 的 X-Ray 開發套件在 Node.js 用戶端中檢測適用於 JavaScript 的 AWS 開發套件。 每當您使用經檢測的用戶端呼叫下游 AWS 服務 或資源時，開發套件會在子區段中記錄有關呼叫的資訊。 AWS 服務 而您在服務中存取的資源會在追蹤地圖上顯示為下游節點，以協助您識別錯誤並調節個別連線的問題。

適用於 Node.js 的 X-Ray 開發套件也提供對 HTTP Web APIs和 SQL 查詢的下游呼叫檢測。將您的 HTTP 用戶端包裝在軟體開發套件的擷取方法中，來記錄傳出 HTTP 呼叫的相關資訊。針對 SQL 用戶端，請使用您資料庫類型的擷取方法。

中介軟體會將抽樣規則套用到傳入請求，判斷要追蹤的請求。您可以設定適用於 Node.js 的 X-Ray 開發套件來調整取樣行為，或記錄應用程式執行所在 AWS 運算資源的相關資訊。

使用註釋與中繼資料，記錄應用程式所做的請求和工作等其他資訊。註釋是簡單的鍵/值對，系統會為其建立索引以用於篩選條件表達式，因此您可以搜尋包含特定資料的追蹤。中繼資料項目的限制性較低，可以記錄整個物件和陣列，任何可以序列化為 JSON 的項目。

> **標註與中繼資料**
>
> 註釋和中繼資料是您使用 X-Ray SDK 新增至區段的任意文字。註釋會編製索引，以便與篩選條件表達式搭配使用。中繼資料不會編製索引，但可以使用 X-Ray 主控台或 API 在原始區段中檢視。您授予 X-Ray 讀取存取權的任何人都可以檢視此資料。

當程式碼中有很多經過檢測的用戶端時，單一請求區段可能包含大量子區段，每個使用經檢測用戶端進行的呼叫都有一個子區段。您可以將用戶端呼叫包裝在自訂子區段中，以組織和群組子區段。您可以為整個函數或任何部分的程式碼建立自訂子區段，並記錄子區段上的中繼資料和註釋，而不必寫入父區段上的所有項目。

如需開發套件類別和方法的參考文件，請參閱適用於 AWS X-Ray Node.js 的開發套件 API 參考。

要求

適用於 Node.js 的 X-Ray 開發套件需要 Node.js 和下列程式庫：

- `atomic-batcher` – 1.0.2
- `cls-hooked` – 4.2.2
- `pkginfo` – 0.4.0
- `semver` – 5.3.0

軟體開發套件會在您使用 NPM 安裝它時提取這些程式庫。

若要追蹤 AWS SDK 用戶端，適用於 Node.js 的 X-Ray 開發套件需要 Node.js 中適用於 JavaScript 的 AWS 開發套件最低版本。

- `aws-sdk` – 2.7.15

相依性管理

適用於 Node.js 的 X-Ray 開發套件可從 NPM 取得。

- 套件 – aws-xray-sdk

針對本機開發，請在您的專案目錄中使用 npm 安裝軟體開發套件。

```
~/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` 選項來在您應用程式的 `package.json` 中將軟體開發套件做為依存項目儲存。

```
~/nodejs-xray$ npm install aws-xray-sdk --save
aws-xray-sdk@3.3.3
```

如果您的應用程式有任何相依性，其版本與 X-Ray SDK 的相依性衝突，則將安裝這兩個版本以確保相容性。如需詳細資訊，請參閱官方 NPM 文件以取得相依性解析。

Node.js 樣本

使用適用於 Node.js 的 AWS X-Ray SDK，end-to-end檢視。

- GitHub 上的 Node.js 範例應用程式。

設定適用於 Node.js 的 X-Ray 開發套件

您可以使用外掛程式設定適用於 Node.js 的 X-Ray 開發套件，以包含應用程式執行的服務相關資訊、修改預設抽樣行為，或新增適用於特定路徑請求的抽樣規則。

章節

服務外掛程式

使用 `plugins` 記錄託管您應用程式之服務的相關資訊。

外掛程式

- Amazon EC2 – `EC2Plugin` 新增執行個體 ID、可用區域和 CloudWatch Logs 群組。
- Elastic Beanstalk – `ElasticBeanstalkPlugin` 新增環境名稱、版本標籤和部署 ID。
- Amazon ECS – `ECSPlugin` 新增容器 ID。

若要使用外掛程式，請使用 `config`方法設定適用於 Node.js 用戶端的 X-Ray 開發套件。

Example app.js - 外掛程式

```
var AWSXRay = require('aws-xray-sdk');
AWSXRay.config([AWSXRay.plugins.EC2Plugin,AWSXRay.plugins.ElasticBeanstalkPlugin]);
```

軟體開發套件也會使用外掛程式設定來設定區段上的 `origin` 欄位。這表示執行您應用程式 AWS 的資源類型。當您使用多個外掛程式時，開發套件會使用下列解析順序來判斷原始伺服器：ElasticBeanstalk > EKS > ECS > EC2。

抽樣規則

SDK 會使用您在 X-Ray 主控台中定義的抽樣規則來決定要記錄哪些請求。預設規則每秒追蹤第一個請求，以及所有傳送追蹤至 X-Ray 服務的任何其他請求的 5%。在 X-Ray 主控台中建立其他規則，以自訂為每個應用程式記錄的資料量。

軟體開發套件會依定義自訂規則的順序進行套用。如果請求符合多個自訂規則，軟體開發套件只會套用第一個規則。

> **Note**
>
> 如果開發套件無法達到 X-Ray 以取得取樣規則，它會每秒還原為第一個請求的預設本機規則，以及每個主機任何額外請求的 5%。如果主機沒有呼叫取樣 APIs許可，或無法連線至 X-Ray 協助程式，而該常駐程式可做為 SDK 進行 API 呼叫的 TCP 代理，則可能會發生這種情況。

您也可以設定 SDK 以從 JSON 文件載入取樣規則。開發套件可以使用本機規則作為 X-Ray 取樣不可用的備份，或僅使用本機規則。

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
  }
}
```

此範例會定義一個自訂規則和預設規則。自訂規則會套用 5% 的取樣率，沒有追蹤 下路徑的最低請求數/api/move/。預設規則會追蹤每秒的第一個請求和 10% 的額外請求。

在本機定義規則的缺點是，固定目標是由記錄器的每個執行個體獨立套用，而不是由 X-Ray 服務管理。當您部署更多主機時，固定速率會乘以，使得難以控制記錄的資料量。

在 上 AWS Lambda，您無法修改取樣率。如果您的函數是由 受檢測的服務呼叫，則 Lambda 會記錄產生該服務取樣請求的呼叫。如果啟用主動追蹤，且不存在追蹤標頭，Lambda 會做出抽樣決策。

若要設定備份規則，請指示適用於 Node.js 的 X-Ray 開發套件從具有 的檔案載入取樣規則`setSamplingRules`。

Example app.js - 檔案的抽樣規則

```
var AWSXRay = require('aws-xray-sdk');
AWSXRay.middleware.setSamplingRules('sampling-rules.json');
```

您也可以在程式碼中定義規則，然後將其做為物件傳遞給 `setSamplingRules`。

Example app.js - 物件的抽樣規則

```
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);
```

若要僅使用本機規則，請呼叫 `disableCentralizedSampling`。

```
AWSXRay.middleware.disableCentralizedSampling()
```

日誌

若要記錄軟體開發套件的輸出，請呼叫 `AWSXRay.setLogger(logger)`，其中 `logger` 是提供標準記錄日誌方法的物件 (`warn`、`info` 等)。

根據預設，軟體開發套件會使用主控台物件上的標準方法，將錯誤訊息記錄到主控台。您可以使用 `AWS_XRAY_DEBUG_MODE`或 `AWS_XRAY_LOG_LEVEL`環境變數來設定內建記錄器的日誌層級。如需有效日誌層級值的清單，請參閱環境變數。

如果您想要為日誌提供不同的格式或目的地，您可以提供軟體開發套件自己的記錄器界面實作，如下所示。您可以使用任何實作此界面的物件。這表示許多日誌程式庫，例如 Winston，都可以直接使用並傳遞至 SDK。

Example app.js - 日誌記錄

```
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]);
```

請在執行其他組態方法前呼叫 `setLogger`，確保擷取來自這些操作的輸出。

X-Ray 協助程式地址

如果 X-Ray 協助程式監聽 以外的連接埠或主機`127.0.0.1:2000`，您可以設定適用於 Node.js 的 X-Ray 開發套件，將追蹤資料傳送至不同的地址。

```
AWSXRay.setDaemonAddress('host:port');
```

您可以透過名稱或 IPv4 地址指定主機。

Example app.js - 精靈地址

```
var AWSXRay = require('aws-xray-sdk');
AWSXRay.setDaemonAddress('daemonhost:8082');
```

若您設定精靈針對 TCP 和 UDP 接聽不同連接埠，您可以在精靈地址設定中同時指定它們。

Example app.js - 位於不同連接埠的精靈地址

```
var AWSXRay = require('aws-xray-sdk');
AWSXRay.setDaemonAddress('tcp:daemonhost:8082 udp:daemonhost:8083');
```

您也可以透過使用 AWS_XRAY_DAEMON_ADDRESS 環境變數來設定精靈地址。

環境變數

您可以使用環境變數來設定適用於 Node.js 的 X-Ray 開發套件。軟體開發套件支援以下變數。

- AWS_XRAY_CONTEXT_MISSING – 設定為 RUNTIME_ERROR 以在未開啟區段時，檢測程式碼嘗試記錄資料時擲回例外狀況。

  有效值

  - RUNTIME_ERROR – 捨棄執行時間例外狀況。
  - LOG_ERROR – 記錄錯誤並繼續 （ 預設 ）。
  - IGNORE_ERROR – 忽略錯誤並繼續。

  當您嘗試在未開啟請求時執行的啟動程式碼中，或在產生新執行緒的程式碼中，使用經檢測的用戶端時，可能會發生與缺少區段或子區段相關的錯誤。

- AWS_XRAY_DAEMON_ADDRESS – 設定 X-Ray 協助程式接聽程式的主機和連接埠。依預設，軟體開發套件會使用 127.0.0.1:2000 進行追蹤資料 (UDP) 和取樣 (TCP)。如果您已設定協助程式在不同的連接埠上接聽，或在不同的主機上執行，請使用此變數。

  格式

  - 相同連接埠 – *address*:*port*
  - 不同的連接埠 – tcp:*address*:*port* udp:*address*:*port*

- AWS_XRAY_DEBUG_MODE – 將 設定為 TRUE，以設定 SDK 將日誌輸出到 主控台的debug關卡。
- AWS_XRAY_LOG_LEVEL – 設定預設記錄器的日誌層級。有效值為debug、info、warn、error、silent。當 AWS_XRAY_DEBUG_MODE 模式設定為 TRUE 時，會忽略此值。
- AWS_XRAY_TRACING_NAME – 設定 SDK 用於區段的服務名稱。覆寫您在 Express 中介軟體上設定的區段名稱。

使用適用於 Node.js 的 X-Ray 開發套件追蹤傳入請求

您可以使用適用於 Node.js 的 X-Ray 開發套件來追蹤 Express and Restify 應用程式在 Amazon EC2EC2 AWS Elastic Beanstalk或 Amazon ECS 中的 EC2 執行個體上提供的傳入 HTTP 請求。

適用於 Node.js 的 X-Ray 開發套件為使用 Express 和 Restify 架構的應用程式提供中介軟體。當您將 X-Ray 中介軟體新增至應用程式時，適用於 Node.js 的 X-Ray 開發套件會為每個抽樣請求建立區段。此區段包括時間、方法，以及 HTTP 請求的處置方式。其他檢測會在此區段上建立子區段。

> **Note**
>
> 對於 AWS Lambda 函數，Lambda 會為每個抽樣請求建立區段。如需更多資訊，請參閱AWS Lambda 而且 AWS X-Ray。

每個客群都有一個名稱，可在服務映射中識別您的應用程式。區段可以靜態命名，或者您可以設定 SDK，根據傳入請求中的主機標頭動態命名。動態命名可讓您根據請求中的網域名稱來分組追蹤，並在名稱不符合預期模式時套用預設名稱（例如，如果主機標頭是偽造的）。

> **轉送的請求**
>
> 如果負載平衡器或其他中介裝置轉送請求到您的應用程式，X-Ray 會從請求中的 `X-Forwarded-For`標頭取得用戶端 IP，而不是從 IP 封包中的來源 IP 取得用戶端 IP。為轉送請求記錄的用戶端 IP 可能是偽造的，因此不應受信任。

轉送請求時，軟體開發套件會在區段中設定額外的欄位來指出這一點。如果區段包含`x_forwarded_for`設為 的欄位`true`，則用戶端 IP 會從 HTTP 請求中的 `X-Forwarded-For`標頭取得。

訊息處理常式會使用 `http` 區塊為每個傳入的請求建立區段，其中包含以下資訊：

- HTTP 方法 – GET、POST、PUT、DELETE 等。
- 用戶端地址 – 傳送請求之用戶端的 IP 地址。
- 回應碼 – 已完成請求的 HTTP 回應碼。
- 時間 – 開始時間（收到請求的時間）和結束時間（傳送回應的時間）。
- 使用者代理程式 — 來自請求`user-agent`的 。
- 內容長度 — `content-length`來自回應的 。

章節

- 使用 Express 追蹤傳入的請求
- 使用 Restify 追蹤傳入的請求
- 設定區段命名策略

使用 Express 追蹤傳入的請求

若要使用 Express 中介軟體，請初始化軟體開發套件用戶端，並使用油 `express.openSegment` 函數傳回的中介軟體，再定義您的路由。

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());
```

定義路由之後，請使用 的輸出`express.closeSegment`，如圖所示，來處理適用於 Node.js 的 X-Ray 開發套件傳回的任何錯誤。

使用 Restify 追蹤傳入的請求

若要使用 Restify 中介軟體，請初始化軟體開發套件用戶端並執行 `enable`。將它您的 Restify 伺服器和區段名稱傳遞給它。

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');
});
```

設定區段命名策略

AWS X-Ray 使用服務名稱來識別您的應用程式，並將其與應用程式使用的其他應用程式、資料庫、外部 APIs AWS 和資源區分開來。當 X-Ray SDK 為傳入請求產生區段時，它會在區段的名稱欄位中記錄應用程式的服務名稱。

X-Ray SDK 可以在 HTTP 請求標頭中的主機名稱之後命名區段。不過，此標頭可能是偽造的，這可能會導致服務映射中出現非預期的節點。若要防止 SDK 因具有偽造主機標頭的請求而不正確命名區段，您必須指定傳入請求的預設名稱。

如果您的應用程式為多個網域提供請求，您可以將 SDK 設定為使用動態命名策略，以在區段名稱中反映這一點。動態命名策略允許 SDK 將主機名稱用於符合預期模式的請求，並將預設名稱套用至不符合的請求。

例如，您可能有一個應用程式向三個子網域提供請求：`www.example.com`、 `api.example.com`和 `static.example.com`。您可以使用動態命名策略搭配 模式`*.example.com`，以不同名稱識別每個子網域的區段，導致服務映射上有三個服務節點。如果您的應用程式收到主機名稱不符合模式的請求，您會在服務映射上看到具有您指定之備用名稱的第四個節點。

若要為所有請求區段使用相同的名稱，請如上一節所述，在初始化中介軟體時指定您應用程式的名稱。

> **Note**
>
> 您可以使用 `AWS_XRAY_TRACING_NAME` 環境變數來覆寫您在程式碼中定義的預設服務名稱。

動態命名策略可定義主機名稱應相符的模式，以及如果 HTTP 請求中的主機名稱不符合模式時要使用的預設名稱。若要動態命名區段，請使用 `AWSXRay.middleware.enableDynamicNaming`。

Example app.js - 動態區段名稱

如果請求中的主機名稱符合 `*.example.com` 模式，請使用該主機名稱。否則，請使用 `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());
```

使用適用於 Node.js 的 X-Ray AWS 開發套件追蹤 SDK 呼叫

當您的應用程式呼叫 AWS 服務 來存放資料、寫入佇列或傳送通知時，適用於 Node.js 的 X-Ray 開發套件會在子區段中追蹤下游的呼叫。您在這些服務中存取的追蹤 AWS 服務和資源 （例如 Amazon S3 儲存貯體或 Amazon SQS 佇列） 會在 X-Ray 主控台的追蹤地圖上顯示為下游節點。

檢測您透過 適用於 JavaScript 的 AWS SDK V2 或 適用於 JavaScript 的 AWS SDK V3 建立的 AWS SDK 用戶端。每個 AWS SDK 版本都提供用於檢測 AWS SDK 用戶端的不同方法。

> **Note**
>
> 目前，與檢測 V2 用戶端相比，適用於 Node.js 的 AWS X-Ray SDK 在檢測 適用於 JavaScript 的 AWS SDK V3 用戶端時傳回的區段資訊較少。例如，代表對 DynamoDB 呼叫的子區段不會傳回資料表名稱。如果您在追蹤中需要此區段資訊，請考慮使用 適用於 JavaScript 的 AWS SDK V2。

適用於 JavaScript 的 AWS SDK V2

您可以在呼叫 時包裝`aws-sdk`您的需求陳述式，以檢測所有 AWS SDK V2 用戶端`AWSXRay.captureAWS`。

Example app.js - AWS SDK 檢測

```
const AWS = AWSXRay.captureAWS(require('aws-sdk'));
```

若要檢測個別用戶端，請在呼叫 時包裝 AWS SDK 用戶端`AWSXRay.captureAWSClient`。例如，若要檢測 AmazonDynamoDB 用戶端：

Example app.js - DynamoDB 用戶端檢測

```
    const AWSXRay = require('aws-xray-sdk');
...
    const ddb = AWSXRay.captureAWSClient(new AWS.DynamoDB());
```

> **Warning**
>
> 請勿將 `captureAWS` 和 `captureAWSClient` 一起使用。這會導致重複的子區段。

如果您想要使用 TypeScript 搭配 ECMAScript 模組 (ESM) 載入JavaScript程式碼，請使用下列範例來匯入程式庫：

Example app.js - AWS SDK 檢測

```
import * as AWS from 'aws-sdk';
import * as AWSXRay from 'aws-xray-sdk';
```

若要使用 ESM 檢測所有 AWS 用戶端，請使用下列程式碼：

Example app.js - AWS SDK 檢測

```
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();
```

對於所有 服務，您可以在 X-Ray 主控台中查看名為 的 API 名稱。對於服務子集，X-Ray SDK 會將資訊新增至區段，以在服務映射中提供更精細的 。

例如，當您使用經檢測的 DynamoDB 用戶端進行呼叫時，軟體開發套件會將資料表名稱新增至以資料表為目標的呼叫區段。在 主控台中，每個資料表都會在服務映射中顯示為個別節點，並具有一般 DynamoDB 節點，用於非以資料表為目標的呼叫。

Example 呼叫 DynamoDB 以儲存項目的子區段

```
{
  "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",
  }
}
```

您存取具名資源時，對以下服務的呼叫會在服務地圖中建立額外節點。未針對特定資源的呼叫，則會建立服務的一般節點。

- Amazon DynamoDB – 資料表名稱
- Amazon Simple Storage Service – 儲存貯體和金鑰名稱
- Amazon Simple Queue Service – 佇列名稱

適用於 JavaScript 的 AWS SDK V3

適用於 JavaScript 的 AWS SDK V3 是模組化的，因此您的程式碼只會載入所需的模組。因此，由於 V3 不支援 `captureAWS`方法，因此無法檢測所有 AWS SDK 用戶端。

如果您想要使用 TypeScript 搭配 ECMAScript 模組 (ESM) 載入 JavaScript 程式碼，您可以使用下列範例來匯入程式庫：

```
import * as AWS from 'aws-sdk';
import * as AWSXRay from 'aws-xray-sdk';
```

使用 `AWSXRay.captureAWSv3Client`方法檢測每個 AWS SDK 用戶端。例如，若要檢測 `AmazonDynamoDB` 用戶端：

Example app.js - 使用適用於 Javascript V3 的 SDK 進行 DynamoDB 用戶端檢測

```
    const AWSXRay = require('aws-xray-sdk');
    const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
...
    const ddb = AWSXRay.captureAWSv3Client(new DynamoDBClient({ region:
 "region" }));
```

使用 適用於 JavaScript 的 AWS SDK V3 時，資料表名稱、儲存貯體和金鑰名稱或佇列名稱等中繼資料目前不會傳回，因此追蹤映射不會像使用 適用於 JavaScript 的 AWS SDK V2 檢測 AWS SDK 用戶端時一樣包含每個具名資源的離散節點。

Example 使用 適用於 JavaScript 的 AWS SDK V3 時，呼叫 DynamoDB 以儲存項目的子區段

```
{
  "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",
  }
}
```

使用適用於 Node.js 的 X-Ray 開發套件追蹤對下游 HTTP Web 服務的呼叫

當您的應用程式呼叫微服務或公有 HTTP APIs，您可以使用適用於 Node.js 的 X-Ray 開發套件用戶端來檢測這些呼叫，並將 API 做為下游服務新增至服務圖表。

將 `http`或 `https`用戶端傳遞至適用於 Node.js 的 X-Ray 開發套件`captureHTTPs`方法，以追蹤傳出呼叫。

> **Note**
>
> 使用協力廠商 HTTP 要求程式庫 (例如 Axios 或 Superagent) 的呼叫會透過 `captureHTTPsGlobal()` API 支援，並在使用原生 `http` 模組時，仍會追蹤這些呼叫。

Example app.js - HTTP 用戶端

```
var AWSXRay = require('aws-xray-sdk');
var http = AWSXRay.captureHTTPs(require('http'));
```

若要啟用所有 HTTP 用戶端上的追蹤，請在載入 `http` 前呼叫 `captureHTTPsGlobal`。

Example app.js - HTTP 用戶端 (全域)

```
var AWSXRay = require('aws-xray-sdk');
AWSXRay.captureHTTPsGlobal(require('http'));
var http = require('http');
```

當您檢測對下游 Web API 的呼叫時，適用於 Node.js 的 X-Ray 開發套件會記錄子區段，其中包含 HTTP 請求和回應的相關資訊。X-Ray 使用 子區段來產生遠端 API 的推斷區段。

Example 下游 HTTP 呼叫的子區段

```
{
  "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 呼叫的推斷區段

```
{
  "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
}
```

使用適用於 Node.js 的 X-Ray 開發套件追蹤 SQL 查詢

在對應的適用於 Node.js 的 X-Ray 開發套件用戶端方法中包裝 SQL 用戶端，以檢測 SQL 資料庫查詢。

- 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);
```

當您使用受檢測用戶端進行 SQL 查詢時，適用於 Node.js 的 X-Ray 開發套件會在子區段中記錄連線及查詢的相關資訊。

在 SQL 子區段中包含其他資料

您可以將其他資訊新增至針對 SQL 查詢產生的子區段，只要它對應到允許清單的 SQL 欄位即可。例如，若要在子區段中記錄已淨化的 SQL 查詢字串，您可以將其直接新增至子區段的 SQL 物件。

Example 將 SQL 指派給子區段

```
    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;
}
```

如需允許清單 SQL 欄位的完整清單，請參閱《 AWS X-Ray 開發人員指南》中的 SQL 查詢。

使用適用於 Node.js 的 X-Ray 開發套件產生自訂子區段

子區段會擴展追蹤的區段，其中包含為處理請求而完成之工作的詳細資訊。每次您與經檢測的用戶端進行呼叫時，X-Ray 開發套件都會記錄子區段中產生的資訊。您可以建立其他子區段來將其他子區段分組、測量程式碼區段的效能，或記錄註釋和中繼資料。

自訂快速子區段

若要為呼叫下游服務的函數建立自訂子區段，請使用 `captureAsyncFunc` 函數。

Example app.js - 快速自訂子區段

```
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();
};
```

在此範例中，應用程式會為針對 sendRequest 函數的呼叫建立名為 send 的自訂子區段。captureAsyncFunc 會在其發出的非同步呼叫完成時傳遞您必須在回撥函數中關閉的子區段。

針對同步函數，您可以使用 captureFunc 函數，自動在函數區塊完成執行後關閉子區段。

當您在區段或其他子區段中建立子區段時，適用於 Node.js 的 X-Ray 開發套件會為其產生 ID，並記錄開始時間和結束時間。

Example 使用中繼資料的子區段

```
"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"
    }
  },
```

自訂 Lambda 子區段

軟體開發套件設定為在偵測到在 Lambda 中執行時自動建立預留位置外觀區段。若要建立基本子區段，其將在 X-Ray 追蹤地圖上建立單一AWS::Lambda::Function節點，請呼叫 並重新利用外觀區

段。如果您使用新 ID 手動建立新區段 (在共用追蹤 ID、父系 ID 和抽樣決策時)，您將能夠傳送新區段。

Example app.js - 手動自訂子區段

```
const segment = AWSXRay.getSegment(); //returns the facade segment
const subsegment = segment.addNewSubsegment('subseg');
...
subsegment.close();
//the segment is closed by the SDK automatically
```

使用適用於 Node.js 的 X-Ray 開發套件將註釋和中繼資料新增至區段

您可以使用註釋和中繼資料記錄有關請求、環境或應用程式的其他資訊。您可以將註釋和中繼資料新增至 X-Ray SDK 建立的區段，或新增至您建立的自訂子區段。

註釋是具有字串、數字或布林值的鍵值對。註釋會編製索引，以便與篩選條件表達式搭配使用。使用標記記錄您想要用來在主控台將追蹤分組的資料，或是在呼叫 `GetTraceSummaries` API 時使用標記。

中繼資料是索引鍵/值對，可以具有任何類型的值，包括物件和清單，但不會編製索引以用於篩選條件表達式。使用中繼資料記錄您希望儲存在追蹤中的其他資料，但不需要搭配搜尋使用。

除了註釋和中繼資料，您也可以在區段上記錄使用者 ID 字串。區段會將使用者 ID 記錄在單獨的欄位中，並建立索引以用於搜尋。

章節

- 使用適用於 Node.js 的 X-Ray 開發套件記錄註釋
- 使用適用於 Node.js 的 X-Ray 開發套件記錄中繼資料
- 使用適用於 Node.js 的 X-Ray 開發套件記錄使用者 IDs

使用適用於 Node.js 的 X-Ray 開發套件記錄註釋

針對您想要建立索引以用於搜尋的區段或子區段，請使用標註來記錄這些區段上的資訊。

註釋要求

- 金鑰 – X-Ray 註釋的金鑰最多可有 500 個英數字元。您不能使用點或句號以外的空格或符號 (。)
- 值 – X-Ray 註釋的值最多可有 1，000 個 Unicode 字元。
- 註釋數量 – 每個追蹤最多可以使用 50 個註釋。

記錄標註

1. 取得目前區段或子區段的參考。

```
var AWSXRay = require('aws-xray-sdk');
...
var document = AWSXRay.getSegment();
```

2. 使用字串索引鍵、布林值、數字或字串值，呼叫 addAnnotation。

```
document.addAnnotation("mykey", "my value");
```

下列範例顯示如何使用包含點的putAnnotation字串索引鍵和布林值、數字或字串值來呼叫 。

```
document.putAnnotation("testkey.test", "my value");
```

軟體開發套件會將標註以鍵/值對記錄在區段文件中的 annotations 物件內。若使用相同鍵呼叫 addAnnotation 兩次，則會覆寫之前在相同區段或子區段上記錄的值。

若要尋找具有特定值註釋的追蹤，請在篩選條件表達式中使用 annotation[*key*]關鍵字。

Example app.js - 標註

```
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) {
...
```

使用適用於 Node.js 的 X-Ray 開發套件記錄中繼資料

針對您不想要建立索引以用於搜尋的區段，請使用中繼資料來記錄這些區段或子區段上的資訊。中繼資料值可以是字串、數字、布林值，或可序列化為 JSON 物件或陣列的任何其他物件。

記錄中繼資料

1. 取得目前區段或子區段的參考。

   ```
   var AWSXRay = require('aws-xray-sdk');
   ...
   var document = AWSXRay.getSegment();
   ```

2. 使用字串鍵、布林值、數字、字串或物件值，以及字串命名空間，呼叫 addMetadata。

   ```
   document.addMetadata("my key", "my value", "my namespace");
   ```

   或

   只使用鍵和值呼叫 addMetadata。

   ```
   document.addMetadata("my key", "my value");
   ```

若您沒有指定命名空間，軟體開發套件會使用 default。若使用相同鍵呼叫 addMetadata 兩次，則會覆寫之前在相同區段或子區段上記錄的值。

使用適用於 Node.js 的 X-Ray 開發套件記錄使用者 IDs

記錄請求區段上的使用者 ID 以識別傳送請求的使用者。此操作與 AWS Lambda 函數不相容，因為 Lambda 環境中的區段不可變。setUser 呼叫只可套用至區段，而非子區段。

記錄使用者 ID

1. 取得目前區段或子區段的參考。

   ```
   var AWSXRay = require('aws-xray-sdk');
   ```

```
...
var document = AWSXRay.getSegment();
```

2. 使用傳送請求之使用者的字串 ID 呼叫 `setUser()`。

```
var user = 'john123';

AWSXRay.getSegment().setUser(user);
```

您可以呼叫 `setUser`，以在 Express 應用程式開始處理請求時馬上記錄使用者 ID。如果您只要使用區段來設定使用者 ID，可以將呼叫鏈結為單行。

Example app.js - 使用者 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) {
...
```

若要尋找使用者 ID 的追蹤，請在篩選條件表達式中使用 `user`關鍵字。

使用 Python

有兩種方式可以檢測您的 Python 應用程式，以將追蹤傳送至 X-Ray：

- AWS Distro for OpenTelemetry Python – 透過 AWS Distro for OpenTelemetry Collector， AWS 提供一組開放原始碼程式庫，用於將相關指標和追蹤傳送至多個 AWS 監控解決方案，包括 Amazon CloudWatch AWS X-Ray和 Amazon OpenSearch Service。
- AWS X-Ray 適用於 Python 的 SDK – 一組程式庫，用於透過 X-Ray 協助程式產生和傳送追蹤至 X-Ray。

如需詳細資訊，請參閱選擇 AWS Distro for OpenTelemetry 和 X-Ray SDKs。

AWS Distro for OpenTelemetry Python

使用 AWS Distro for OpenTelemetry (ADOT) Python，您可以檢測您的應用程式一次，並將相關指標和追蹤傳送至多個 AWS 監控解決方案，包括 Amazon CloudWatch AWS X-Ray和 Amazon OpenSearch Service。搭配 ADOT 使用 X-Ray 需要兩個元件：啟用 OpenTelemetry SDK 以搭配 X-Ray 使用，以及啟用 AWS Distro for OpenTelemetry Collector 以搭配 X-Ray 使用。ADOT Python 包含自動檢測支援，讓您的應用程式無需變更程式碼即可傳送追蹤。

若要開始使用，請參閱 AWS Distro for OpenTelemetry Python 文件。

如需搭配 AWS AWS X-Ray 和其他 使用 Distro for OpenTelemetry 的詳細資訊 AWS 服務，請參閱 AWS Distro for OpenTelemetry 或 AWS Distro for OpenTelemetry 文件。

如需語言支援和用量的詳細資訊，請參閱 AWS GitHub 上的可觀測性。

AWS X-Ray 適用於 Python 的 SDK

適用於 Python 的 X-Ray 開發套件是 Python Web 應用程式的程式庫，提供產生追蹤資料並將其傳送至 X-Ray 協助程式的類別和方法。追蹤資料包含應用程式所服務傳入 HTTP 請求的相關資訊，以及應用程式使用 AWS SDK、HTTP 用戶端或 SQL 資料庫連接器對下游服務進行的呼叫。您也可以手動建立區段，並將除錯資訊新增至註釋和中繼資料中。

您可以使用 `pip` 下載軟體開發套件。

```
$ pip install aws-xray-sdk
```

> **Note**
>
> 適用於 Python 的 X-Ray 開發套件是開放原始碼專案。您可以關注專案，並在 GitHub 上提交問題與提取 (pull) 請求：github.com/aws/aws-xray-sdk-python

若您使用 Django 或 Flask，請從將軟體開發套件中介軟體新增到您的應用程式開始，來追蹤傳入請求。中介軟體會為每個追蹤的請求建立區段，並在傳送回應時完成區段。當區段開啟時，您可以使用軟體開發套件用戶端的方法，將資訊新增到區段，並建立子區段以追蹤下游呼叫。軟體開發套件也會在區段為開啟時自動記錄應用程式擲回的例外狀況。針對其他應用程式，您可以手動建立區段。

對於由經檢測的應用程式或服務呼叫的 Lambda 函數，Lambda 會自動讀取追蹤標頭並追蹤取樣的請求。對於其他 函數，您可以設定 Lambda 來取樣和追蹤傳入的請求。無論哪種情況，Lambda 都會建立區段，並將其提供給 X-Ray 開發套件。

> **Note**
>
> 在 Lambda 上，X-Ray 開發套件是選用的。如果您未在函數中使用它，您的服務映射仍會包含 Lambda 服務的節點，以及每個 Lambda 函數的一個節點。透過新增 SDK，您可以檢測函數程式碼，將子區段新增至 Lambda 記錄的函數區段。如需更多資訊，請參閱AWS Lambda 而且 AWS X-Ray。

如需在 Lambda 中檢測工作程序的 Python 函數範例，請參閱 。

接著，使用適用於 Python 的 X-Ray 開發套件，透過修補應用程式的程式庫來檢測下游呼叫。軟體開發套件支援以下程式庫。

支援的程式庫

- `botocore`、 `boto3` – 檢測 AWS SDK for Python (Boto) 用戶端。
- `pynamodb` – Instrument PynamoDB 的 Amazon DynamoDB 用戶端版本。
- `aiobotocore`、 `aioboto3` – 檢測適用於 Python 用戶端的 SDK 非同步整合版本。
- `requests`、 `aiohttp` – 檢測高階 HTTP 用戶端。
- `httplib`、 `http.client` – 檢測低階 HTTP 用戶端和使用它們的更高層級程式庫。
- `sqlite3` – 檢測 SQLite 用戶端。
- `mysql-connector-python` – 檢測 MySQL 用戶端。

- pg8000 – Instrument Pure-Python PostgreSQL 介面。
- psycopg2 – Instrument PostgreSQL 資料庫轉接器。
- pymongo – 檢測 MongoDB 用戶端。
- pymysql – 檢測 MySQL 和 MariaDB 的 PyMySQL MySQL 型用戶端。 MariaDB

每當您的應用程式呼叫 AWS SQL 資料庫或其他 HTTP 服務時，開發套件會在子區段中記錄有關呼叫的資訊。 AWS 服務 而您在服務中存取的資源會在追蹤地圖上顯示為下游節點，以協助您識別錯誤並調節個別連線的問題。

開始使用 SDK 之後，請設定記錄器和中介軟體來自訂其行為。您可以新增外掛程式，以記錄執行應用程式所需的運算資源相關資料、定義抽樣規則以自訂抽樣行為，並設定日誌層級以在應用程式日誌中查看更多或更少的軟體開發套件資訊。

使用註釋與中繼資料，記錄應用程式所做的請求和工作等其他資訊。註釋是簡單的鍵/值對，系統會為其建立索引以用於篩選條件表達式，因此您可以搜尋包含特定資料的追蹤。中繼資料項目的限制性較低，可以記錄整個物件和陣列，任何可以序列化為 JSON 的項目。

> **標註與中繼資料**
>
> 註釋和中繼資料是您使用 X-Ray SDK 新增至區段的任意文字。註釋會編製索引，以便與篩選條件表達式搭配使用。中繼資料不會編製索引，但可以使用 X-Ray 主控台或 API 在原始區段中檢視。您授予 X-Ray 讀取存取權的任何人都可以檢視此資料。

當程式碼中有很多經過檢測的用戶端時，單一請求區段可能包含大量子區段，每個使用經檢測用戶端進行的呼叫都有一個子區段。您可以將用戶端呼叫包裝在自訂子區段中，以組織和群組子區段。您可以為整個函數或一部分的程式碼建立自訂子區段。您接著可以在子區段上記錄中繼資料及標註，而非將所有項目寫入父區段。

如需 SDK 類別和方法的參考文件，請參閱AWS X-Ray 適用於 Python 的 SDK API 參考。

要求

適用於 Python 的 X-Ray 開發套件支援下列語言和程式庫版本。

- Python – 2.7、3.4 和更新版本
- Django – 1.10 及更新版本
- Flask – 0.10 及更新版本

- aiohttp – 2.3.0 及更新版本
- AWS SDK for Python (Boto) – 1.4.0 及更新版本
- botocore – 1.5.0 及更新版本
- enum – 0.4.7 及更新版本，適用於 Python 3.4.0 及更新版本
- jsonpickle – 1.0.0 及更新版本
- setuptools - 40.6.3 和更新版本
- wrapt – 1.11.0 及更新版本

相依性管理

適用於 Python 的 X-Ray 開發套件可從 取得`pip`。

- 套件 – `aws-xray-sdk`

在您的 `requirements.txt` 檔案中將軟體開發套件新增為依存項目。

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 部署應用程式，Elastic Beanstalk `requirements.txt`會自動在 中安裝所有套件。

設定適用於 Python 的 X-Ray 開發套件

適用於 Python 的 X-Ray 開發套件具有名為 的類別`xray_recorder`，可提供全域記錄器。您可以設定全域記錄器來自訂為傳入 HTTP 呼叫建立區段的中介軟體。

章節

- 服務外掛程式
- 抽樣規則
- 日誌
- 程式碼中的記錄器組態

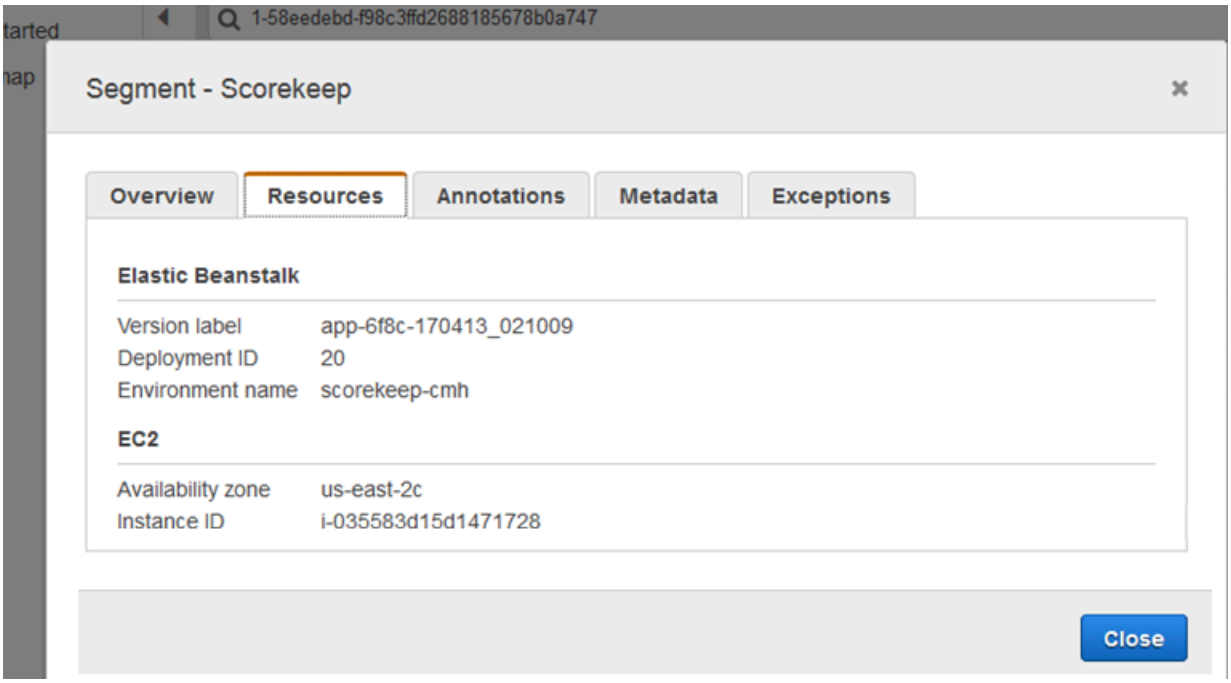
- 使用 Django 的記錄器組態
- 環境變數

服務外掛程式

使用 `plugins` 記錄託管您應用程式之服務的相關資訊。

外掛程式

- Amazon EC2 – `EC2Plugin` 新增執行個體 ID、可用區域和 CloudWatch Logs 群組。
- Elastic Beanstalk – `ElasticBeanstalkPlugin` 新增環境名稱、版本標籤和部署 ID。
- Amazon ECS – `ECSPlugin` 新增容器 ID。



若要使用外掛程式，請在 `xray_recorder` 上呼叫 `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**
>
> 由於 `plugins` 是以元組形式傳入，因此指定單一外掛程式，時請務必包含結尾。例如 `plugins = ('EC2Plugin',)`

您也可以使用優先於程式碼中設定值的環境變數，來設定記錄器。

在修補程式庫前設定外掛程式來記錄下游呼叫。

軟體開發套件也會使用外掛程式設定來設定區段上的 `origin` 欄位。這表示執行您應用程式 AWS 的資源類型。當您使用多個外掛程式時，開發套件會使用下列解析順序來判斷原始伺服器：ElasticBeanstalk > EKS > ECS > EC2。

抽樣規則

SDK 會使用您在 X-Ray 主控台中定義的抽樣規則來決定要記錄哪些請求。預設規則每秒追蹤第一個請求，以及所有傳送追蹤至 X-Ray 服務的任何其他請求的 5%。在 X-Ray 主控台中建立其他規則，以自訂為每個應用程式記錄的資料量。

軟體開發套件會依定義自訂規則的順序進行套用。如果請求符合多個自訂規則，軟體開發套件只會套用第一個規則。

> **Note**
>
> 如果開發套件無法達到 X-Ray 以取得取樣規則，它會每秒還原為第一個請求的預設本機規則，以及每個主機任何額外請求的 5%。如果主機沒有呼叫取樣 APIs許可，或無法連線至 X-Ray 協助程式，而該常駐程式可做為 SDK 進行 API 呼叫的 TCP 代理，則可能會發生這種情況。

您也可以設定 SDK 以從 JSON 文件載入取樣規則。開發套件可以使用本機規則作為 X-Ray 取樣不可用的備份，或僅使用本機規則。

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
  }
}
```

此範例會定義一個自訂規則和預設規則。自訂規則會套用 5% 的取樣率，沒有追蹤 下路徑的最低請求數/api/move/。預設規則會追蹤每秒的第一個請求和 10% 的額外請求。

在本機定義規則的缺點是，固定目標是由記錄器的每個執行個體獨立套用，而不是由 X-Ray 服務管理。當您部署更多主機時，固定速率會乘以，使得難以控制記錄的資料量。

在 上 AWS Lambda，您無法修改取樣率。如果您的函數是由 受檢測的服務呼叫，則 Lambda 會記錄產生該服務取樣請求的呼叫。如果啟用主動追蹤，且不存在追蹤標頭，Lambda 會做出抽樣決策。

若要設定備份抽樣規則，請呼叫 `xray_recorder.configure` (如以下範例所示)，其中 *`rules`* 為規則的字典或指向包含抽樣規則 JSON 檔案的絕對路徑。

```
xray_recorder.configure(sampling_rules=rules)
```

若僅要使用本機規則，請使用 `LocalSampler` 設定記錄器。

```
from aws_xray_sdk.core.sampling.local.sampler import LocalSampler
xray_recorder.configure(sampler=LocalSampler())
```

您也可以設定全域記錄器來停用所有傳入請求的抽樣和檢測。

Example main.py – 停用取樣

```
xray_recorder.configure(sampling=False)
```

日誌

SDK 使用 Python 的內建`logging`模組搭配預設`WARNING`記錄層級。為 `aws_xray_sdk` 類別取得記錄器的參考，然後在其上呼叫 `setLevel` 來為程式庫及您其餘的應用程式設定不同日誌層級。

Example app.py – 記錄

```
logging.basicConfig(level='WARNING')
logging.getLogger('aws_xray_sdk').setLevel(logging.ERROR)
```

當您手動產生子區段時，可使用除錯日誌來識別問題，例如未結束的子區段。

程式碼中的記錄器組態

可以從 xray_recorder 上的 configure 方法取得其他可用設定。

- context_missing – 設定為 LOG_ERROR以避免在未開啟區段時，檢測程式碼嘗試記錄資料時擲出例外狀況。
- daemon_address – 設定 X-Ray 協助程式接聽程式的主機和連接埠。
- service – 設定 SDK 用於區段的服務名稱。
- plugins – 記錄應用程式 AWS 資源的相關資訊。
- sampling – 設定為 False以停用取樣。
- sampling_rules – 設定包含取樣規則的 JSON 檔案路徑。

Example main.py – 停用內容遺失例外狀況

```
from aws_xray_sdk.core import xray_recorder

xray_recorder.configure(context_missing='LOG_ERROR')
```

使用 Django 的記錄器組態

若您使用 Django 框架，您可以使用 Django settings.py 檔案來在全域記錄器上設定選項。

- AUTO_INSTRUMENT （僅限 Django) – 記錄內建資料庫和範本轉譯操作的子區段。
- AWS_XRAY_CONTEXT_MISSING – 設定為 LOG_ERROR以避免在未開啟區段時，檢測程式碼嘗試記錄資料時擲回例外狀況。
- AWS_XRAY_DAEMON_ADDRESS – 設定 X-Ray 協助程式接聽程式的主機和連接埠。
- AWS_XRAY_TRACING_NAME – 設定 SDK 用於區段的服務名稱。
- PLUGINS – 記錄應用程式 AWS 資源的相關資訊。
- SAMPLING – 設定為 False以停用取樣。
- SAMPLING_RULES – 設定包含取樣規則的 JSON 檔案路徑。

若要在 settings.py 中啟用記錄器組態，請將 Django 中介軟體新增至已安裝的應用程式清單。

Example settings.py – 已安裝的應用程式

```
INSTALLED_APPS = [
    ...
    'django.contrib.sessions',
    'aws_xray_sdk.ext.django',
]
```

在名為 XRAY_RECORDER 的字典中設定可用設定。

Example settings.py – 已安裝的應用程式

```
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,
}
```

環境變數

您可以使用環境變數來設定適用於 Python 的 X-Ray SDK。軟體開發套件支援以下變數：

- AWS_XRAY_TRACING_NAME – 設定 SDK 用於區段的服務名稱。覆寫您以程式設計方式設定的服務名稱。
- AWS_XRAY_SDK_ENABLED – 設為 時false， 會停用 SDK。軟體開發套件預設為啟用，除非環境變數設定為 false。
  - 停用時，全域記錄器會自動產生虛擬、不會傳送給協助程式的區段和子區段，且自動修補會停用。中介軟體是編寫做為透過全域記錄器的包裝函式。透過中介軟體產生的所有區段和子區段，也會成為虛擬區段和虛擬子區段。
  - 透過環境變數，或是透過與 aws_xray_sdk 程式庫內 global_sdk_config 物件的直接互動，設定 AWS_XRAY_SDK_ENABLED 的值。環境變數設定會覆寫這些互動。
- AWS_XRAY_DAEMON_ADDRESS – 設定 X-Ray 協助程式接聽程式的主機和連接埠。依預設，軟體開發套件會使用 127.0.0.1:2000 進行追蹤資料 (UDP) 和取樣 (TCP)。如果您已設定協助程式在不同的連接埠上接聽，或在不同的主機上執行，請使用此變數。

格式

- 相同連接埠 – *address*:*port*
- 不同的連接埠 – tcp:*address*:*port* udp:*address*:*port*

- AWS_XRAY_CONTEXT_MISSING – 設定為 RUNTIME_ERROR 以在未開啟區段時，檢測程式碼嘗試記錄資料時擲回例外狀況。

  有效值

  - RUNTIME_ERROR – 捨棄執行時間例外狀況。
  - LOG_ERROR – 記錄錯誤並繼續 （預設）。
  - IGNORE_ERROR – 忽略錯誤並繼續。

  當您嘗試在未開啟請求時執行的啟動程式碼中，或在產生新執行緒的程式碼中，使用經檢測的用戶端時，可能會發生與缺少區段或子區段相關的錯誤。

環境變數會覆寫程式碼中所設的值。

使用適用於 Python 的 X-Ray 開發套件中介軟體追蹤傳入請求

當您將中介軟體新增至應用程式並設定區段名稱時，適用於 Python 的 X-Ray 開發套件會為每個抽樣請求建立區段。此區段包括時間、方法，以及 HTTP 請求的處置方式。其他檢測會在此區段上建立子區段。

適用於 Python 的 X-Ray 開發套件支援下列中介軟體，以檢測傳入的 HTTP 請求：

- Django
- Flask
- Bottle

> **Note**
>
> 對於 AWS Lambda 函數，Lambda 會為每個抽樣請求建立區段。如需更多資訊，請參閱AWS Lambda 而且 AWS X-Ray。

如需在 Lambda 中檢測工作程序的 Python 函數範例，請參閱 。

針對指令碼或其他框架上的 Python 應用程式，您可以手動建立區段。

每個客群都有一個名稱，可在服務映射中識別您的應用程式。區段可以靜態命名，或者您可以設定 SDK，根據傳入請求中的主機標頭動態命名。動態命名可讓您根據請求中的網域名稱來分組追蹤，並在名稱不符合預期模式時套用預設名稱（例如，如果主機標頭是偽造的）。

> **轉送的請求**
>
> 如果負載平衡器或其他中介裝置轉送請求到您的應用程式，X-Ray 會從請求中的 `X-Forwarded-For`標頭取得用戶端 IP，而不是從 IP 封包中的來源 IP 取得用戶端 IP。為轉送請求記錄的用戶端 IP 可能是偽造的，因此不應受信任。

轉送請求時，軟體開發套件會在區段中設定額外的欄位來指出這一點。如果區段包含`x_forwarded_for`設為 的欄位`true`，用戶端 IP 會從 HTTP 請求中的 `X-Forwarded-For`標頭取得。

中介軟體會使用 `http` 區塊為每個傳入的請求建立區段，其中包含以下資訊：

- HTTP 方法 – GET、POST、PUT、DELETE 等。
- 用戶端地址 – 傳送請求之用戶端的 IP 地址。
- 回應碼 – 已完成請求的 HTTP 回應碼。
- 時間 – 開始時間（收到請求的時間）和結束時間（傳送回應的時間）。
- 使用者代理程式 — 來自請求`user-agent`的 。
- 內容長度 — `content-length`來自回應的 。

章節

- 將中介軟體新增至應用程式 (Django)
- 將中介軟體新增至應用程式 (Flask)
- 將中介軟體新增至應用程式 (Bottle)
- 手動檢測 Python 程式碼
- 設定區段命名策略

將中介軟體新增至應用程式 (Django)

將中介軟體新增至您 `settings.py` 檔案中的 `MIDDLEWARE` 清單。X-Ray 中介軟體應為您 `settings.py` 檔案中的第一行，確保會記錄在其他中介軟體中失敗的請求。

Example settings.py - 適用於 Python 中介軟體的 X-Ray 開發套件

```
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'
]
```

將 X-Ray 開發套件 Django 應用程式新增至 `settings.py` 檔案中的`INSTALLED_APPS`清單。這將允許在您的應用程式啟動期間設定 X-Ray 記錄器。

Example settings.py - 適用於 Python Django 的 X-Ray 開發套件應用程式

```
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',
]
```

在您的 `settings.py` 檔案中設定區段名稱。

Example settings.py – 區段名稱

```
XRAY_RECORDER = {
    'AWS_XRAY_TRACING_NAME': 'My application',
    'PLUGINS': ('EC2Plugin',),
}
```

這可讓 X-Ray 記錄器以預設取樣率追蹤 Django 應用程式提供的請求。您可以在您的 Django 設定檔中設定記錄器來套用自訂抽樣規則或變更其他設定。

> **Note**
>
> 由於 `plugins` 是以元組形式傳入，因此指定單一外掛程式，時請務必包含結尾。例如 `plugins = ('EC2Plugin',)`

將中介軟體新增至應用程式 (Flask)

若要檢測您的 Flask 應用程式，請先在 `xray_recorder` 上設定區段名稱。然後，使用 `XRayMiddleware` 函數來在程式碼中修補您的 Flask 應用程式。

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)
```

這會通知 X-Ray 記錄器以預設取樣率追蹤 Flask 應用程式提供的請求。您可以在程式碼中設定記錄器來套用自訂抽樣規則或變更其他設定。

將中介軟體新增至應用程式 (Bottle)

若要檢測您的 Bottle 應用程式，請先在 `xray_recorder` 上設定區段名稱。然後，使用 `XRayMiddleware` 函數來在程式碼中修補您的 Bottle 應用程式。

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))
```

這可讓 X-Ray 記錄器以預設取樣率追蹤 Bottle 應用程式提供的請求。您可以在程式碼中設定記錄器來套用自訂抽樣規則或變更其他設定。

手動檢測 Python 程式碼

若您並未使用 Django 或 Flask，您可以手動建立區段。您可以為每個傳入請求建立區段，或在修補的 HTTP 或 AWS SDK 用戶端周圍建立區段，以提供記錄器新增子區段的內容。

Example main.py – 手動檢測

```
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()
```

設定區段命名策略

AWS X-Ray 使用服務名稱來識別您的應用程式，並將其與應用程式使用的其他應用程式、資料庫、外部 APIs AWS 和資源區分開來。當 X-Ray SDK 為傳入請求產生區段時，它會在區段的名稱欄位中記錄應用程式的服務名稱。

X-Ray SDK 可以在 HTTP 請求標頭中的主機名稱後面命名區段。不過，此標頭可以偽造，這可能會導致服務映射中出現非預期的節點。若要防止 SDK 因具有偽造主機標頭的請求而不正確命名區段，您必須指定傳入請求的預設名稱。

如果您的應用程式為多個網域提供請求，您可以將 SDK 設定為使用動態命名策略，以在區段名稱中反映這一點。動態命名策略可讓 SDK 將主機名稱用於符合預期模式的請求，並將預設名稱套用至不符合的請求。

例如，您可能有一個應用程式向三個子網域提供請求：`www.example.com`、 `api.example.com`和 `static.example.com`。您可以使用動態命名策略搭配 模式`*.example.com`，以不同名稱識別每個

子網域的區段，導致服務映射上有三個服務節點。如果您的應用程式收到主機名稱不符合模式的請求，您會在服務映射上看到具有您指定之備用名稱的第四個節點。

若要為所有請求區段使用相同的名稱，請如上一節所述，在設定記錄器時指定您應用程式的名稱。

動態命名策略可定義主機名稱應相符的模式，以及如果 HTTP 請求中的主機名稱不符合模式時要使用的預設名稱。若要在 Django 中動態命名區段，請將 DYNAMIC_NAMING 設定新增到您的 settings.py 檔案。

Example settings.py – 動態命名

```
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 '*' 來比對任何字串，或是 '?' 來比對任何單一字元。針對 Flask，請在程式碼中設定記錄器。

Example main.py：//www. – 客群名稱

```
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 環境變數來覆寫您在程式碼中定義的預設服務名稱。

修補程式庫來檢測下游呼叫

若要檢測下游呼叫，請使用適用於 Python 的 X-Ray 開發套件來修補應用程式使用的程式庫。適用於 Python 的 X-Ray 開發套件可以修補下列程式庫。

支援的程式庫

- botocore、 boto3 – 檢測 AWS SDK for Python (Boto) 用戶端。
- pynamodb – Instrument PynamoDB 的 Amazon DynamoDB 用戶端版本。

- aiobotocore、 aioboto3 – 檢測適用於 Python 用戶端的 SDK 非同步整合版本。
- requests、 aiohttp – 檢測高階 HTTP 用戶端。
- httplib、 http.client – 檢測低階 HTTP 用戶端和使用它們的更高層級程式庫。
- sqlite3 – 檢測 SQLite 用戶端。
- mysql-connector-python – 檢測 MySQL 用戶端。
- pg8000 – Instrument Pure-Python PostgreSQL 介面。
- psycopg2 – Instrument PostgreSQL 資料庫轉接器。
- pymongo – 檢測 MongoDB 用戶端。
- pymysql – 檢測 MySQL 和 MariaDB 的 PyMySQL MySQL 型用戶端。 MariaDB

當您使用修補的程式庫時，適用於 Python 的 X-Ray 開發套件會為呼叫建立子區段，並記錄請求和回應中的資訊。區段必須透過軟體開發套件中介軟體或 AWS Lambda供軟體開發套件使用，以建立子區段。

> **Note**
>
> 若您使用 SQLAlchemy ORM，您可以透過匯入 SQLAlchemy 工作階段和查詢類別的軟體開發套件版本，來檢測您的 SQL 查詢。如需說明，請參閱使用 SQLAlchemy ORM。

若要修補所有可用程式庫，請使用 `aws_xray_sdk.core` 中的 `patch_all` 函數。有些程式庫 (例如 `httplib` 和 `urllib`) 可能需要呼叫 `patch_all(double_patch=True)` 以啟用雙重修補。

Example main.py – 修補所有支援的程式庫

```
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()
```

若要修補單一程式庫，請以該程式庫名稱的元組來呼叫 `patch`。若要執行這個作業，您將需要提供單一個元素清單。

Example main.py – 修補特定程式庫

```
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**
>
> 在某些情況下，您用來修補程式庫的鍵會與程式庫名稱不相符。有些鍵會做為一或多個程式庫的別名。
>
> 程式庫別名
>
> - `httplib`–`httplib` 和 `http.client`
> - `mysql` – `mysql-connector-python`

追蹤非同步工作的內容

對於`asyncio`整合式程式庫，或為非同步函數建立子區段，您還必須使用非同步內容設定適用於 Python 的 X-Ray 開發套件。匯入 `AsyncContext`類別，並將其執行個體傳遞至 X-Ray 記錄器。

> **Note**
>
> Web 架構支援程式庫，例如 AIOHTTP，將不會經由 `aws_xray_sdk.core.patcher` 模組獲得處理。它們不會出現在支援程式庫的 `patcher` 目錄中。

Example main.py – 修補程式 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)
```

使用適用於 Python 的 X-Ray AWS 開發套件追蹤 SDK 呼叫

當您的應用程式呼叫 AWS 服務 來存放資料、寫入佇列或傳送通知時，適用於 Python 的 X-Ray 開發套件會在子區段中追蹤下游的呼叫。您在這些服務中存取的追蹤 AWS 服務 和資源 （例如 Amazon S3 儲存貯體或 Amazon SQS 佇列），會在 X-Ray 主控台的追蹤地圖上顯示為下游節點。

當您修補程式botocore庫時，適用於 Python 的 X-Ray SDK 會自動檢測所有 AWS SDK 用戶端。您無法檢測個別用戶端。

對於所有 服務，您可以在 X-Ray 主控台中查看名為 的 API 名稱。對於服務子集，X-Ray SDK 會將資訊新增至區段，以在服務映射中提供更精細的 。

例如，當您使用經檢測的 DynamoDB 用戶端進行呼叫時，軟體開發套件會將資料表名稱新增至以資料表為目標的呼叫區段。在 主控台中，每個資料表都會在服務映射中顯示為個別節點，並具有一般 DynamoDB 節點，用於非以資料表為目標的呼叫。

Example 呼叫 DynamoDB 以儲存項目的子區段

```
{
  "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",
```

```
  }
}
```

您存取具名資源時，對以下服務的呼叫會在服務地圖中建立額外節點。未針對特定資源的呼叫，則會建立服務的一般節點。

- Amazon DynamoDB – 資料表名稱
- Amazon Simple Storage Service – 儲存貯體和金鑰名稱
- Amazon Simple Queue Service – 佇列名稱

使用適用於 Python 的 X-Ray 開發套件追蹤對下游 HTTP Web 服務的呼叫

當您的應用程式呼叫微服務或公有 HTTP APIs，您可以使用適用於 Python 的 X-Ray 開發套件來檢測這些呼叫，並將 API 做為下游服務新增至服務圖表。

若要檢測 HTTP 用戶端，請修補您用來進行傳出呼叫的程式庫。若您使用 `requests` 或 Python 內建的 HTTP 用戶端，只需進行這些作業。針對 `aiohttp`，請同時使用非同步內容設定記錄器。

若您使用 `aiohttp` 3 的用戶端 API，您也需要使用軟體開發套件提供的追蹤組態執行個體設定 `ClientSession` 的部分。

Example **`aiohttp`** 3 用戶端 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()
```

當您檢測對下游 Web API 的呼叫時，適用於 Python 的 X-Ray 開發套件會記錄子區段，其中包含 HTTP 請求和回應的相關資訊。X-Ray 使用 子區段來產生遠端 API 的推斷區段。

Example 下游 HTTP 呼叫的子區段

```
{
  "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 呼叫的推斷區段

```
{
  "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
}
```

使用適用於 Python 的 X-Ray 開發套件產生自訂子區段

子區段會延伸追蹤的區段，其中包含為處理請求而完成之工作的詳細資訊。每次您與經檢測的用戶端進行呼叫時，X-Ray 開發套件都會記錄子區段中產生的資訊。您可以建立其他子區段來將其他子區段分組、測量程式碼區段的效能，或記錄註釋和中繼資料。

若要管理子區段，請使用 `begin_subsegment` 和 `end_subsegment` 方法。

Example main.py – 自訂子區段

```
from aws_xray_sdk.core import xray_recorder

subsegment = xray_recorder.begin_subsegment('annotations')
subsegment.put_annotation('id', 12345)
xray_recorder.end_subsegment()
```

若要建立同步函數的子區段，請使用 @xray_recorder.capture 裝飾項目。您可以將子區段的名稱傳遞給擷取函數，或是不傳遞它來使用函數名稱。

Example main.py – 函數子區段

```
from aws_xray_sdk.core import xray_recorder

@xray_recorder.capture('## create_user')
def create_user():
...
```

針對非同步函數，請使用 @xray_recorder.capture_async 裝飾項目，並將非同步內容傳遞給記錄器。

Example main.py – 非同步函數子區段

```
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()
```

當您在區段或其他子區段中建立子區段時，適用於 Python 的 X-Ray 開發套件會為其產生 ID，並記錄開始時間和結束時間。

Example 使用中繼資料的子區段

```
"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"
    }
  },
```

使用適用於 Python 的 X-Ray 開發套件將註釋和中繼資料新增至區段

您可以使用註釋和中繼資料記錄有關請求、環境或應用程式的其他資訊。您可以將註釋和中繼資料新增至 X-Ray 開發套件建立的區段，或新增至您建立的自訂子區段。

註釋是具有字串、數字或布林值的鍵值對。註釋會編製索引，以便與篩選條件表達式搭配使用。使用標記記錄您想要用來在主控台將追蹤分組的資料，或是在呼叫 `GetTraceSummaries` API 時使用標記。

中繼資料是索引鍵/值對，可以具有任何類型的值，包括物件和清單，但不會編製索引以用於篩選條件表達式。使用中繼資料記錄您希望儲存在追蹤中的其他資料，但不需要搭配搜尋使用。

除了註釋和中繼資料，您也可以在區段上記錄使用者 ID 字串。區段會將使用者 ID 記錄在單獨的欄位中，並建立索引以用於搜尋。

章節

- 使用適用於 Python 的 X-Ray 開發套件記錄註釋
- 使用適用於 Python 的 X-Ray 開發套件記錄中繼資料
- 使用適用於 Python 的 X-Ray 開發套件記錄使用者 IDs

使用適用於 Python 的 X-Ray 開發套件記錄註釋

針對您想要建立索引以用於搜尋的區段或子區段，請使用標註來記錄這些區段上的資訊。

註釋要求

- 金鑰 – X-Ray 註釋的金鑰最多可有 500 個英數字元。您不能使用點或句號以外的空格或符號 (。)
- 值 – X-Ray 註釋的值最多可有 1，000 個 Unicode 字元。
- 註釋數量 – 每個追蹤最多可以使用 50 個註釋。

記錄標註

1. 從 xray_recorder 取得目前區段或子區段的參考。

   ```
   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. 使用字串索引鍵、布林值、數字或字串值，呼叫 put_annotation。

   ```
   document.put_annotation("mykey", "my value");
   ```

   下列範例顯示如何使用包含點的putAnnotation字串索引鍵和布林值、數字或字串值來呼叫 。

   ```
   document.putAnnotation("testkey.test", "my value");
   ```

或者，您可以使用 xray_recorder 上的 put_annotation 方法。此方法會在目前的子區段或區段 (若沒有任何開啟的子區段) 上記錄標註。

```
xray_recorder.put_annotation("mykey", "my value");
```

軟體開發套件會將標註以鍵/值對記錄在區段文件中的 annotations 物件內。若使用相同鍵呼叫 put_annotation 兩次，則會覆寫之前在相同區段或子區段上記錄的值。

若要尋找具有特定值註釋的追蹤，請在篩選條件表達式中使用 annotation[*key*]關鍵字。

使用適用於 Python 的 X-Ray 開發套件記錄中繼資料

針對您不想要建立索引以用於搜尋的區段，請使用中繼資料來記錄這些區段或子區段上的資訊。中繼資料值可以是字串、數字、布林值，或可序列化為 JSON 物件或陣列的任何物件。

記錄中繼資料

1. 從 xray_recorder 取得目前區段或子區段的參考。

   ```
   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. 使用字串鍵、布林值、數字、字串或物件值，以及字串命名空間，呼叫 put_metadata。

   ```
   document.put_metadata("my key", "my value", "my namespace");
   ```

   或

   只使用鍵和值呼叫 put_metadata。

   ```
   document.put_metadata("my key", "my value");
   ```

或者，您可以使用 xray_recorder 上的 put_metadata 方法。此方法會在目前的子區段或區段 (若沒有任何開啟的子區段) 上記錄中繼資料。

```
xray_recorder.put_metadata("my key", "my value");
```

若您沒有指定命名空間，軟體開發套件會使用 default。若使用相同鍵呼叫 put_metadata 兩次，則會覆寫之前在相同區段或子區段上記錄的值。

使用適用於 Python 的 X-Ray 開發套件記錄使用者 IDs

記錄請求區段上的使用者 ID 以識別傳送請求的使用者。

記錄使用者 ID

1. 從 xray_recorder 取得目前區段的參考。

```
from aws_xray_sdk.core import xray_recorder
...
document = xray_recorder.current_segment()
```

2. 使用傳送請求之使用者的字串 ID 呼叫 `setUser`。

```
document.set_user("U12345");
```

您可以在控制器中呼叫 `set_user`，以在應用程式開始處理請求時馬上記錄使用者 ID。

若要尋找使用者 ID 的追蹤，請在篩選條件表達式中使用 `user`關鍵字。

檢測部署在無伺服器環境中的 Web 框架

適用於 Python 的 AWS X-Ray 開發套件支援檢測部署在無伺服器應用程式中的 Web 架構。無伺服器解決方案是雲端原生架構，能讓您將更多的操作責任轉移到 AWS，提高您的敏捷度和創新能力。

無伺服器架構是一種軟體應用程式模型，可讓您建置和執行應用程式和服務，而完全不用考慮伺服器。這種做法可免除基礎設施管理工作，例如伺服器或叢集佈建、修補、作業系統維護和容量佈建。您可以為幾乎任何應用程式類型或後端服務建立無伺服器解決方案，並為您包辦執行和擴展高可用性應用程式所需的一切工作。

本教學課程說明如何自動 AWS X-Ray 檢測部署到無伺服器環境的 Web 架構，例如 Flask 或 Django。應用程式的 X-Ray 檢測可讓您檢視從 Amazon API Gateway 透過您的 AWS Lambda 函數進行的所有下游呼叫，以及應用程式進行的外撥呼叫。

適用於 Python 的 X-Ray SDK 支援下列 Python 應用程式架構：

- Flask 版本 0.8 或更新版本
- Django 版本 1.0 或更新版本

本教學課程會開發範例無伺服器應用程式，部署至 Lambda 並由 API Gateway 叫用。本教學課程使用 Zappa 將應用程式自動部署到 Lambda，並設定 API Gateway 端點。

先決條件

- Zappa
- Python – 2.7 或 3.6 版。

- AWS CLI – 確認您的 AWS CLI 已使用 AWS 區域 您要部署應用程式的 帳戶設定。
- Pip
- Virtualenv

步驟 1：建立環境

在此步驟中，您將建立使用 `virtualenv` 來主控應用程式的虛擬環境。

1. 使用 AWS CLI為應用程式建立目錄。然後，切換至新目錄。

```
mkdir serverless_application
cd serverless_application
```

2. 接下來，在您的新目錄中建立虛擬環境。使用下列命令，將其啟動。

```
# Create our virtual environment
virtualenv serverless_env

# Activate it
source serverless_env/bin/activate
```

3. 將 X-Ray、Flask、Zappa 和請求程式庫安裝到您的環境。

```
# Install X-Ray, Flask, Zappa, and Requests into your environment
pip install aws-xray-sdk flask zappa requests
```

4. 將應用程式程式碼新增到 `serverless_application` 目錄。在這個範例中，我們可以建置出 Flasks 的 Hello World 範例。

   在 `serverless_application` 目錄中，建立名為 `my_app.py` 的檔案。然後，使用文字編輯器來新增下列命令。這個應用程式會檢測請求程式庫、修補 Flask 應用程式的中介軟體，並開啟端點 `'/'`。

```
# 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
```

步驟 2：建立和部署 Zappa 環境

在此步驟中，您將使用 Zappa 自動設定 API Gateway 端點，然後部署到 Lambda。

1. 從 `serverless_application` 目錄當中起始 Zappa。在這個範例中，我們使用的是預設設定，但如果您有自訂偏好設定，則 Zappa 會顯示組態指示。

```
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. 啟用 X-Ray。開啟 `zappa_settings.json` 檔案，並確認其類似此範例。

```
{
```

```
    "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. 將 `"xray_tracing": true` 新增做為組態檔的項目：

```
{
    "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. 部署應用程式。這會自動設定 API Gateway 端點，並將您的程式碼上傳至 Lambda。

```
zappa deploy
```

```
...
Deploying API Gateway..
Deployment complete!: https://**********.execute-api.us-west-2.amazonaws.com/dev
```

步驟 3：啟用 API Gateway 的 X-Ray 追蹤

在此步驟中，您將與 API Gateway 主控台互動，以啟用 X-Ray 追蹤。

1. 登入 AWS Management Console 並開啟 API Gateway 主控台，網址為 https：//https://console.aws.amazon.com/apigateway/。
2. 尋找新產生的 API。該項目看起來應該會類似 `serverless-exam-dev`。
3. 選擇 Stages (階段)。

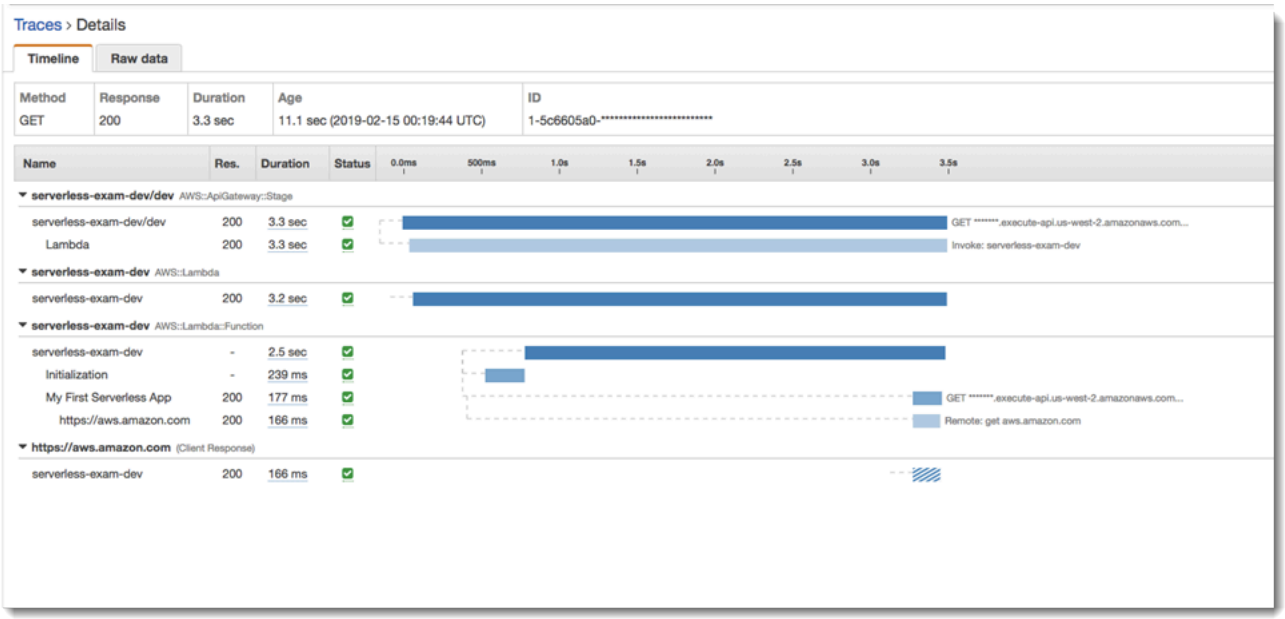
4. 選擇您的部署階段名稱。預設值為 dev。
5. 在 Logs/Tracing (日誌/追蹤) 標籤中，選取 Enable X-Ray Tracing (啟用 X-Ray 追蹤) 方塊。
6. 選擇 Save Changes (儲存變更)。
7. 在您的瀏覽器中存取端點。如果您使用的是 Hello World 應用程式範例，這時應該顯示如下。

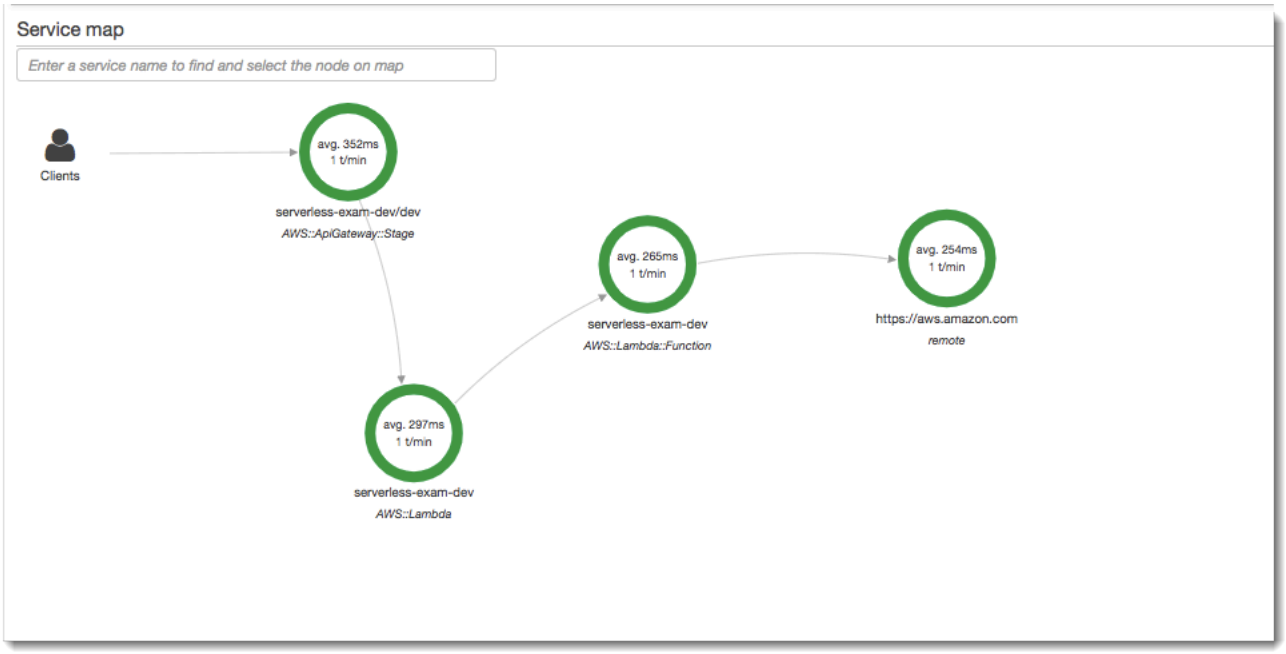
```
"Hello, World: https://aws.amazon.com/"
```

步驟 4：檢視已建立的追蹤

在此步驟中，您將與 X-Ray 主控台互動，以檢視範例應用程式建立的追蹤。如需追蹤分析的詳細演練，請參閱檢視服務映射。

1. 登入 AWS Management Console ，並在 https：//https://console.aws.amazon.com/xray/home 開啟 X-Ray 主控台。
2. 檢視 API Gateway、Lambda 函數和 Lambda 容器所產生的區段。
3. 在 Lambda 函數區段下，檢視名為 的子區段My First Serverless App。它的後面是一個名為 https://aws.amazon.com 的第二個子區段。
4. 在初始化期間，Lambda 也可能產生名為 的第三個子區段initialization。





步驟 5：清除

務必終止您不再使用的資源，避免累積超出預期的費用。在此教學示範中，像是 Zappa 等工具會簡化無伺服器解決方案的重新部署。

若要從 Lambda、API Gateway 和 Amazon S3 移除您的應用程式，請使用 在專案目錄中執行下列命令 AWS CLI。

```
zappa undeploy dev
```

後續步驟

透過新增 AWS 用戶端並使用 X-Ray 檢測，將更多功能新增至您的應用程式。進一步了解 Serverless on 的無 AWS伺服器運算選項。

使用 .NET

有兩種方式可以檢測您的 .NET 應用程式，以將追蹤傳送至 X-Ray：

- AWS Distro for OpenTelemetry .NET – 透過 AWS Distro for OpenTelemetry Collector AWS X-Ray， AWS 提供一組開放原始碼程式庫，用於將相關指標和追蹤傳送至多個 AWS 監控解決方案，包括 Amazon CloudWatch 和 Amazon OpenSearch Service。
- AWS X-Ray 適用於 .NET 的 SDK – 一組程式庫，用於透過 X-Ray 協助程式產生和傳送追蹤至 X-Ray。

如需詳細資訊，請參閱選擇 AWS Distro for OpenTelemetry 和 X-Ray SDKs。

AWS Distro for OpenTelemetry .NET

使用 AWS Distro for OpenTelemetry .NET，您可以檢測您的應用程式一次，並將相關指標和追蹤傳送至多個 AWS 監控解決方案，包括 Amazon CloudWatch AWS X-Ray和 Amazon OpenSearch Service。搭配 AWS Distro for OpenTelemetry 使用 X-Ray 需要兩個元件：啟用 OpenTelemetry SDK 以搭配 X-Ray 使用，以及啟用 AWS Distro for OpenTelemetry Collector 以搭配 X-Ray 使用。

若要開始使用，請參閱 AWS Distro for OpenTelemetry .NET 文件。

如需搭配 AWS AWS X-Ray 和其他 使用 Distro for OpenTelemetry 的詳細資訊 AWS 服務，請參閱 AWS Distro for OpenTelemetry 或 AWS Distro for OpenTelemetry 文件。

如需語言支援和用量的詳細資訊，請參閱 AWS GitHub 上的可觀測性。

AWS X-Ray 適用於 .NET 的 SDK

適用於 .NET 的 X-Ray 開發套件是用於檢測 C# .NET Web 應用程式、.NET Core Web 應用程式和 .NET Core 函數的程式庫 AWS Lambda。它提供產生追蹤資料並將其傳送至 X-Ray 協助程式的類別和方法。這包括應用程式所提供服務的傳入請求，以及應用程式對下游、HTTP Web APIs AWS 服務和 SQL 資料庫發出的呼叫的相關資訊。

> **Note**
>
> 適用於 .NET 的 X-Ray 開發套件是開放原始碼專案。您可以追蹤專案，並在 GitHub 上提交問題與提取請求：github.com/aws/aws-xray-sdk-dotnet

針對 Web 應用程式，請先將訊息處理常式新增至 Web 組態以追蹤傳入的請求。訊息處理常式會為每個追蹤的請求建立區段，並在傳送回應時完成區段。當區段開啟時，您可以使用軟體開發套件用戶端的方法，將資訊新增到區段，並建立子區段以追蹤下游呼叫。軟體開發套件也會在區段為開啟時自動記錄應用程式擲回的例外狀況。

對於由經檢測的應用程式或服務呼叫的 Lambda 函數，Lambda 會自動讀取追蹤標頭並追蹤取樣的請求。對於其他 函數，您可以設定 Lambda 來取樣和追蹤傳入的請求。無論哪種情況，Lambda 都會建立區段，並將其提供給 X-Ray 開發套件。

> **Note**
>
> 在 Lambda 上，X-Ray 開發套件是選用的。如果您未在函數中使用它，您的服務映射仍會包含 Lambda 服務的節點，以及每個 Lambda 函數的一個節點。透過新增 SDK，您可以檢測函數程式碼，將子區段新增至 Lambda 記錄的函數區段。如需更多資訊，請參閱AWS Lambda 而且 AWS X-Ray。

接著，使用適用於 .NET 的 X-Ray 開發套件來檢測您的 適用於 .NET 的 AWS SDK 用戶端。每當您使用經檢測的用戶端呼叫下游 AWS 服務 或資源時，開發套件會在子區段中記錄有關呼叫的資訊。 AWS 服務以及您在服務內存取的資源會在追蹤地圖上顯示為下游節點，以協助您識別錯誤並調節個別連線的問題。

適用於 .NET 的 X-Ray 開發套件也提供對 HTTP Web APIs和 SQL 資料庫進行下游呼叫的檢測。適用於 `System.Net.HttpWebRequest` 的 `GetResponseTraced` 延伸方法可追蹤傳出的 HTTP 呼叫。您可以使用適用於 .NET 的 X-Ray 開發套件的 版本`SqlCommand`來檢測 SQL 查詢。

開始使用 SDK 之後，請設定記錄器和訊息處理常式來自訂其行為。您可以新增外掛程式，以記錄執行應用程式所需的運算資源相關資料、定義抽樣規則以自訂抽樣行為，並設定日誌層級以在應用程式日誌中查看更多或更少的軟體開發套件資訊。

使用註釋與中繼資料，記錄應用程式所做的請求和工作等其他資訊。註釋是簡單的鍵/值對，系統會為其建立索引以用於篩選條件表達式，因此您可以搜尋包含特定資料的追蹤。中繼資料項目的限制性較低，可以記錄整個物件和陣列，任何可以序列化為 JSON 的項目。

> **標註與中繼資料**
>
> 註釋和中繼資料是您使用 X-Ray SDK 新增至區段的任意文字。註釋會編製索引，以便與篩選條件表達式搭配使用。中繼資料不會編製索引，但可以使用 X-Ray 主控台或 API 在原始區段中檢視。您授予 X-Ray 讀取存取權的任何人都可以檢視此資料。

當程式碼中有許多經過檢測的用戶端時，單一請求區段可能包含大量子區段，每個使用經檢測用戶端進行的呼叫都有一個子區段。您可以將用戶端呼叫包裝在自訂子區段中，以組織和群組子區段。您可以為整個函數或任何部分的程式碼建立自訂子區段，並記錄子區段上的中繼資料和註釋，而不必寫入父區段上的所有項目。

如需軟體開發套件之類別和方法的參考文件，請參閱以下項目：

- AWS X-Ray 適用於 .NET 的 SDK API 參考
- AWS X-Ray SDK for .NET Core API 參考

相同套件可同時支援 .NET 和 .NET Core，但使用的類別不同。除非類別是 .NET Core 專屬，否則本章範例都會連結至 .NET API 參考。

要求

適用於 .NET 的 X-Ray 開發套件需要 .NET Framework 4.5 或更新版本，以及 適用於 .NET 的 AWS SDK。

若是 .NET Core 應用程式和函數，則軟體開發套件需要 .NET Core 2.0 或更新版本。

將適用於 .NET 的 X-Ray 開發套件新增至您的應用程式

使用 NuGet 將適用於 .NET 的 X-Ray 開發套件新增至您的應用程式。

在 Visual Studio 中安裝適用於 .NET 的 X-Ray 開發套件與 NuGet 套件管理員

1. 選擇 Tools (工具)、NuGet Package Manager (NuGet 套件管理員)、Manage NuGet Packages for Solution (管理解決方案的 NuGet 套件)。
2. 搜尋 AWSXRayRecorder。
3. 選擇套件，然後選擇 Install (安裝)。

相依性管理

適用於 .NET 的 X-Ray 開發套件可從 Nuget 取得。使用套件管理員安裝 SDK：

```
Install-Package AWSXRayRecorder -Version 2.10.1
```

nuget `AWSXRayRecorder v2.10.1` 套件具有下列相依性：

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)
```

如需相依性管理的詳細資訊，請參閱 Microsoft 有關 Nuget 相依性和 Nuget 相依性解析的文件。

設定適用於 .NET 的 X-Ray 開發套件

您可以使用外掛程式設定適用於 .NET 的 X-Ray 開發套件，以包含應用程式執行的服務相關資訊、修改預設抽樣行為，或新增適用於特定路徑請求的抽樣規則。

針對 .NET web 應用程式，請將鍵新增至您 `Web.config` 檔案中的 `appSettings` 區段。

Example Web.config

```
<configuration>
  <appSettings>
    <add key="AWSXRayPlugins" value="EC2Plugin"/>
    <add key="SamplingRuleManifest" value="sampling-rules.json"/>
  </appSettings>
</configuration>
```

針對 .NET Core，請建立名為 `appsettings.json` 的檔案，其中帶有名為 `XRay` 的最上層鍵。

Example .NET appsettings.json

```
{
  "XRay": {
    "AWSXRayPlugins": "EC2Plugin",
    "SamplingRuleManifest": "sampling-rules.json"
  }
}
```

然後，在您的應用程式程式碼中，建置組態物件並使用它來初始化 X-Ray 記錄器。請在初始化記錄器前執行此作業。

Example .NET Core Program.cs – 記錄器組態

```
using Amazon.XRay.Recorder.Core;
...
AWSXRayRecorder.InitializeInstance(configuration);
```

如果您要檢測 .NET Core Web 應用程式，也可以在設定訊息處理常式時，將組態物件傳遞至 `UseXRay`方法。對於 Lambda 函數，請使用 `InitializeInstance`方法，如上所示。

如需 .NET Core 組態 API 的詳細資訊，請參閱在 ASP.NET：// 上設定 Amazon Core 應用程式。docs.microsoft.com

章節

- 外掛程式
- 抽樣規則
- 記錄日誌 (.NET)
- 記錄日誌 (.NET Core)
- 環境變數

外掛程式

使用外掛程式來新增託管您應用程式的服務相關資料。

外掛程式

- Amazon EC2 – `EC2Plugin` 新增執行個體 ID、可用區域和 CloudWatch Logs 群組。
- Elastic Beanstalk – `ElasticBeanstalkPlugin` 新增環境名稱、版本標籤和部署 ID。
- Amazon ECS – `ECSPlugin` 新增容器 ID。

若要使用外掛程式，請新增 `AWSXRayPlugins`設定，以設定適用於 .NET 用戶端的 X-Ray 開發套件。若要將多個外掛程式套用至您的應用程式，請在相同設定中指定它們，並以逗號分隔。

Example Web.config - 外掛程式

```
<configuration>
```

```
  <appSettings>
    <add key="AWSXRayPlugins" value="EC2Plugin,ElasticBeanstalkPlugin"/>
  </appSettings>
</configuration>
```

Example .NET Core appsettings.json – 外掛程式

```
{
  "XRay": {
    "AWSXRayPlugins": "EC2Plugin,ElasticBeanstalkPlugin"
  }
}
```

抽樣規則

SDK 會使用您在 X-Ray 主控台中定義的抽樣規則來決定要記錄哪些請求。預設規則每秒追蹤第一個請求，以及所有傳送追蹤至 X-Ray 服務的任何其他請求的 5%。在 X-Ray 主控台中建立其他規則，以自訂為每個應用程式記錄的資料量。

軟體開發套件會依定義自訂規則的順序進行套用。如果請求符合多個自訂規則，軟體開發套件只會套用第一個規則。

> **Note**
>
> 如果開發套件無法達到 X-Ray 以取得取樣規則，它會每秒還原為第一個請求的預設本機規則，以及每個主機任何額外請求的 5%。如果主機沒有呼叫取樣 APIs許可，或無法連線至 X-Ray 協助程式，而該常駐程式可做為 SDK 進行 API 呼叫的 TCP 代理，則可能會發生這種情況。

您也可以設定 SDK 以從 JSON 文件載入取樣規則。開發套件可以使用本機規則作為 X-Ray 取樣不可用的備份，或僅使用本機規則。

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
  }
}
```

此範例會定義一個自訂規則和預設規則。自訂規則會套用 5% 的取樣率，沒有追蹤 下路徑的最低請求數/api/move/。預設規則會追蹤每秒的第一個請求和 10% 的額外請求。

在本機定義規則的缺點是，固定目標是由記錄器的每個執行個體獨立套用，而不是由 X-Ray 服務管理。當您部署更多主機時，固定速率會乘以，使得難以控制記錄的資料量。

在 上 AWS Lambda，您無法修改取樣率。如果您的函數是由 受檢測的服務呼叫，則 Lambda 會記錄產生該服務取樣請求的呼叫。如果啟用主動追蹤，且不存在追蹤標頭，Lambda 會做出抽樣決策。

若要設定備份規則，請指示適用於 .NET 的 X-Ray 開發套件使用 SamplingRuleManifest設定從檔案載入取樣規則。

Example .NET Web.config - 抽樣規則

```
<configuration>
  <appSettings>
    <add key="SamplingRuleManifest" value="sampling-rules.json"/>
  </appSettings>
</configuration>
```

Example .NET Core appsettings.json – 取樣規則

```
{
  "XRay": {
    "SamplingRuleManifest": "sampling-rules.json"
  }
}
```

若僅要使用本機規則，請使用 LocalizedSamplingStrategy 建置記錄器。若您已設定備份規則，請移除該組態。

Example .NET global.asax – 本機取樣規則

```
var recorder = new AWSXRayRecorderBuilder().WithSamplingStrategy(new
 LocalizedSamplingStrategy("samplingrules.json")).Build();
AWSXRayRecorder.InitializeInstance(recorder: recorder);
```

Example .NET Core Program.cs – 本機取樣規則

```
var recorder = new AWSXRayRecorderBuilder().WithSamplingStrategy(new
 LocalizedSamplingStrategy("sampling-rules.json")).Build();
AWSXRayRecorder.InitializeInstance(configuration,recorder);
```

記錄日誌 (.NET)

適用於 .NET 的 X-Ray 開發套件使用與 相同的記錄機制適用於 .NET 的 AWS SDK。如果您已將應用程式設定為記錄 適用於 .NET 的 AWS SDK 輸出，則相同的組態會套用至適用於 .NET 的 X-Ray 開發套件輸出。

若要設定記錄日誌，請將名為 `aws` 的組態區段新增至您的 `App.config` 檔案或 `Web.config` 檔案。

Example Web.config - 記錄

```
...
<configuration>
  <configSections>
    <section name="aws" type="Amazon.AWSSection, AWSSDK.Core"/>
  </configSections>
  <aws>
    <logging logTo="Log4Net"/>
  </aws>
</configuration>
```

如需詳細資訊，請參閱《 適用於 .NET 的 AWS SDK 開發人員指南》中的設定您的 適用於 .NET 的 AWS SDK 應用程式。

記錄日誌 (.NET Core)

適用於 .NET 的 X-Ray 開發套件使用與 相同的記錄選項適用於 .NET 的 AWS SDK。若要設定 .NET Core 應用程式的記錄，請將記錄選項傳遞至 `AWSXRayRecorder.RegisterLogger`方法。

例如，若要使用 log4net，請建立定義記錄器、輸出格式及檔案位置的組態檔。

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>
```

然後建立記錄器並在您的程式程式碼中套用組態。

Example .NET Core Program.cs：//www.healthnet：/www.healthnet：//www.net Core； -

```
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)
  {
  ...
  }
}
```

如需設定 log4net 的詳細資訊，請參閱 logging.apache.org 上的組態。

環境變數

您可以使用環境變數來設定適用於 .NET 的 X-Ray 開發套件。軟體開發套件支援以下變數。

- AWS_XRAY_TRACING_NAME – 設定 SDK 用於區段的服務名稱。覆寫您在 servlet 篩選條件的區段命名策略中設定的服務名稱。
- AWS_XRAY_DAEMON_ADDRESS – 設定 X-Ray 協助程式接聽程式的主機和連接埠。依預設，軟體開發套件會使用 127.0.0.1:2000 進行追蹤資料 (UDP) 和取樣 (TCP)。如果您已設定協助程式在不同的連接埠上接聽，或正在不同的主機上執行，請使用此變數。

  格式

  - 相同連接埠 – *address*:*port*
  - 不同的連接埠 – tcp:*address*:*port* udp:*address*:*port*
- AWS_XRAY_CONTEXT_MISSING – 設定為 RUNTIME_ERROR 以在未開啟區段時，檢測程式碼嘗試記錄資料時擲回例外狀況。

  有效值

  - RUNTIME_ERROR – 捨棄執行時間例外狀況。
  - LOG_ERROR – 記錄錯誤並繼續 （預設）。
  - IGNORE_ERROR – 忽略錯誤並繼續。

  當您嘗試在未開啟請求時執行的啟動程式碼中，或在產生新執行緒的程式碼中，使用經檢測的用戶端時，可能會發生與缺少區段或子區段相關的錯誤。

使用適用於 .NET 的 X-Ray 開發套件檢測傳入 HTTP 請求

您可以使用 X-Ray 開發套件來追蹤您的應用程式在 Amazon EC2EC2 AWS Elastic Beanstalk或 Amazon ECS 中的 EC2 執行個體上提供的傳入 HTTP 請求。

使用訊息處理常式檢測傳入的 HTTP 請求。當您將 X-Ray 訊息處理常式新增至應用程式時，適用於 .NET 的 X-Ray 開發套件會為每個取樣請求建立區段。此區段包括時間、方法，以及 HTTP 請求的處置方式。其他檢測會在此區段上建立子區段。

> **Note**
>
> 對於 AWS Lambda 函數，Lambda 會為每個抽樣請求建立區段。如需更多資訊，請參閱AWS Lambda 而且 AWS X-Ray。

每個客群都有一個名稱，可在服務映射中識別您的應用程式。區段可以靜態命名，或者您可以設定 SDK，根據傳入請求中的主機標頭動態命名。動態命名可讓您根據請求中的網域名稱來分組追蹤，並在名稱不符合預期模式時套用預設名稱（例如，如果主機標頭是偽造的）。

> **轉送的請求**
>
> 如果負載平衡器或其他中介裝置轉送請求到您的應用程式，X-Ray 會從請求中的 `X-Forwarded-For`標頭取得用戶端 IP，而不是從 IP 封包中的來源 IP 取得用戶端 IP。為轉送請求記錄的用戶端 IP 可能是偽造的，因此不應受信任。

訊息處理常式會使用 `http` 區塊為每個傳入的請求建立區段，其中包含以下資訊：

- HTTP 方法 – GET、POST、PUT、DELETE 等。
- 用戶端地址 – 傳送請求之用戶端的 IP 地址。
- 回應碼 – 已完成請求的 HTTP 回應碼。
- 時間 – 開始時間（收到請求的時間）和結束時間（傳送回應的時間）。
- 使用者代理程式 — 來自請求`user-agent`的 。
- 內容長度 — `content-length`來自回應的 。

章節

- 檢測傳入的請求 (.NET)
- 檢測傳入的請求 (.NET Core)
- 設定區段命名策略

檢測傳入的請求 (.NET)

若要檢測您應用程式處理的請求，請在您 `global.asax` 檔案的 `Init` 方法中呼叫 `RegisterXRay`。

Example global.asax - 訊息處理常式

```
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");
    }
  }
}
```

檢測傳入的請求 (.NET Core)

若要檢測應用程式提供的請求，請在啟動類別UseXRay方法中的任何其他中介軟體之前呼叫 Configure 方法，因為理想情況下，X-Ray 中介軟體應該是處理請求的第一個中介軟體，以及處理管道中回應的最後一個中介軟體。

> **Note**
>
> 對於 .NET Core 2.0，如果您在應用程式中有UseExceptionHandler方法，請務必在UseExceptionHandler方法UseXRay後呼叫 ，以確保記錄例外狀況。

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 方法也可以接收組態物件做為第二個引數。

```
app.UseXRay("MyApp", configuration);
```

設定區段命名策略

AWS X-Ray 使用服務名稱來識別您的應用程式，並將其與應用程式使用的其他應用程式、資料庫、外部 APIs AWS 和資源區分開來。當 X-Ray SDK 為傳入請求產生區段時，它會在區段的名稱欄位中記錄應用程式的服務名稱。

X-Ray SDK 可以在 HTTP 請求標頭中的主機名稱後面命名區段。不過，此標頭可以偽造，這可能會導致服務映射中出現非預期的節點。若要防止 SDK 因具有偽造主機標頭的請求而不正確命名區段，您必須指定傳入請求的預設名稱。

如果您的應用程式為多個網域提供請求，您可以將 SDK 設定為使用動態命名策略，以在區段名稱中反映這一點。動態命名策略可讓 SDK 將主機名稱用於符合預期模式的請求，並將預設名稱套用至不符合的請求。

例如，您可能有一個應用程式向三個子網域提供請求：www.example.com、 api.example.com和 static.example.com。您可以使用動態命名策略搭配 模式*.example.com，以不同名稱識別每個子網域的區段，導致服務映射上有三個服務節點。如果您的應用程式收到主機名稱不符合模式的請求，您會在服務映射上看到具有您指定之備用名稱的第四個節點。

若要為所有請求區段使用相同的名稱，請如上一節所述，在初始化訊息處理常式時指定您應用程式的名稱。這與建立 FixedSegmentNamingStrategy 並將其傳遞給 RegisterXRay 方法有相同的效果。

```
AWSXRayASPNET.RegisterXRay(this, new FixedSegmentNamingStrategy("MyApp"));
```

> **Note**
>
> 您可以使用 AWS_XRAY_TRACING_NAME 環境變數來覆寫您在程式碼中定義的預設服務名稱。

動態命名策略可定義主機名稱應相符的模式，以及如果 HTTP 請求中的主機名稱不符合模式時要使用的預設名稱。若要動態命名區段，請建立 DynamicSegmentNamingStrategy 並將其傳遞至 RegisterXRay 方法。

```
AWSXRayASPNET.RegisterXRay(this, new DynamicSegmentNamingStrategy("MyApp",
 "*.example.com"));
```

使用適用於 AWS .NET 的 X-Ray 開發套件追蹤 SDK 呼叫

當您的應用程式呼叫 AWS 服務 來存放資料、寫入佇列或傳送通知時，適用於 .NET 的 X-Ray 開發套件會在子區段中追蹤下游的呼叫。您在這些服務中存取的追蹤 AWS 服務 和資源 （ 例如 Amazon S3 儲存貯體或 Amazon SQS 佇列 ），會在 X-Ray 主控台的追蹤地圖上顯示為下游節點。

您可以在建立用戶端RegisterXRayForAllServices之前先呼叫 來檢測所有 適用於 .NET 的 AWS SDK 用戶端。

Example SampleController.cs - DynamoDB 用戶端檢測

```
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;
```

```
    });
```

若要針對某些特定服務檢測用戶端，請呼叫 RegisterXRay，而不是 RegisterXRayForAllServices。將醒目提示的文字取代為服務的用戶端界面名稱。

```
AWSSDKHandler.RegisterXRay<IAmazonDynamoDB>()
```

對於所有 服務，您可以在 X-Ray 主控台中查看名為 的 API 名稱。對於服務子集，X-Ray SDK 會將資訊新增至區段，以在服務映射中提供更精細的 。

例如，當您使用經檢測的 DynamoDB 用戶端進行呼叫時，軟體開發套件會將資料表名稱新增至以資料表為目標的呼叫區段。在 主控台中，每個資料表都會在服務映射中顯示為個別節點，並具有一般 DynamoDB 節點，用於非以資料表為目標的呼叫。

Example 呼叫 DynamoDB 以儲存項目的子區段

```
{
  "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",
  }
}
```

您存取具名資源時，對以下服務的呼叫會在服務地圖中建立額外節點。未針對特定資源的呼叫，則會建立服務的一般節點。

- Amazon DynamoDB – 資料表名稱
- Amazon Simple Storage Service – 儲存貯體和金鑰名稱

- Amazon Simple Queue Service – 佇列名稱

使用適用於 .NET 的 X-Ray 開發套件追蹤下游 HTTP Web 服務的呼叫

當您的應用程式呼叫微服務或公有 HTTP APIs 時，您可以使用適用於 .NET 的 X-Ray 開發套件`GetResponseTraced`延伸方法`System.Net.HttpWebRequest`，來檢測這些呼叫，並將 API 做為下游服務新增至服務圖表。

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();
}
```

針對非同步呼叫，請使用 `GetAsyncResponseTraced`。

```
request.GetAsyncResponseTraced();
```

若您使用 system.net.http.httpclient，請使用 `HttpClientXRayTracingHandler` 委派處理常式來記錄呼叫。

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);
}
```

當您檢測對下游 Web API 的呼叫時，適用於 .NET 的 X-Ray 開發套件會記錄子區段，其中包含 HTTP 請求和回應的相關資訊。X-Ray 使用 子區段來產生 API 的推斷區段。

Example 下游 HTTP 呼叫的子區段

```
{
  "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 呼叫的推斷區段

```
{
  "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
```

```
}
```

使用適用於 .NET 的 X-Ray 開發套件追蹤 SQL 查詢

適用於 .NET 的 X-Ray 開發套件為 `System.Data.SqlClient.SqlCommand`提供名為 的包裝函式類別`TraceableSqlCommand`，您可以用來取代 `SqlCommand`。您可以使用 `TraceableSqlCommand` 類別來初始化 SQL 命令。

使用同步和非同步方法追蹤 SQL 查詢

以下範例說明如何使用 `TraceableSqlCommand` 來以同步和非同步方式自動追蹤 SQL Server 查詢。

Example **`Controller.cs`** - SQL 用戶端檢測 (同步)

```
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，以非同步方式執行查詢。

Example **`Controller.cs`** - SQL 用戶端檢測 (非同步)

```
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();
  }
}
```

收集對 SQL Server 執行的 SQL 查詢

您可以啟用 SqlCommand.CommandText 的擷取，做為 SQL 查詢所建立子區段的一部分。SqlCommand.CommandText 會在子區段 JSON 中顯示為欄位 sanitized_query。基於安全考量，此功能預設為停用。

> **Note**
>
> 如果您在 SQL 查詢中包含純文字形式的敏感資訊，請勿啟用收集功能。

您可以透過兩種方式啟用收集 SQL 查詢：

- 針對您的應用程式，在全域組態中將 CollectSqlQueries 屬性設定為 true。
- 將 TraceableSqlCommand 執行個體中的 collectSqlQueries 參數設為 true，以收集執行個體內的呼叫。

啟用全域 CollectSqlQueries 屬性

以下範例說明如何啟用適用於 .NET 和 .NET Core 的 CollectSqlQueries 屬性。

.NET

若要在 .NET 中將您應用程式全域組態的 CollectSqlQueries 屬性設為 true，請修改您 App.config 或 Web.config 檔案的 appsettings，如下所示。

Example **App.config** 或者 **Web.config** – 全域啟用 SQL 查詢集合

```
<configuration>
<appSettings>
    <add key="CollectSqlQueries" value="true">
</appSettings>
</configuration>
```

.NET Core

若要在 .NET Core `true` 中將應用程式全域組態中的 `CollectSqlQueries` 屬性設定為 ，請在 X-Ray 金鑰下修改 `appsettings.json` 檔案，如下所示。

Example **`appsettings.json`** – 全域啟用 SQL 查詢集合

```
{
  "XRay": {
    "CollectSqlQueries":"true"
  }
}
```

啟用 collectSqlQueries 參數

您可以將 `TraceableSqlCommand` 執行個體中的 `collectSqlQueries` 參數設為 `true`，以收集使用該執行個體進行之 SQL Server 查詢的 SQL 查詢文字。將參數設為 `false` 會停用 `TraceableSqlCommand` 執行個體的 `CollectSqlQuery` 功能。

> **Note**
>
> `TraceableSqlCommand` 執行個體中的 `collectSqlQueries` 值會覆寫 `CollectSqlQueries` 屬性之全域組態中設定的值。

Example 範例 **`Controller.cs`** – 啟用執行個體的 SQL 查詢集合

```
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();
  }
```

```
}
```

建立其他子區段

子區段會延伸追蹤的區段，其中包含為處理請求而完成之工作的詳細資訊。每次您與經檢測的用戶端進行呼叫時，X-Ray 開發套件都會記錄子區段中產生的資訊。您可以建立其他子區段來將其他子區段分組、測量程式碼區段的效能，或記錄註釋和中繼資料。

若要管理子區段，請使用 BeginSubsegment 和 EndSubsegment 方法。執行 try 區塊之子區段中的任何工作，並使用 AddException 追蹤例外狀況。呼叫 finally 區塊中的 EndSubsegment，以確保子區段結束。

Example Controller.cs – 自訂子區段

```
AWSXRayRecorder.Instance.BeginSubsegment("custom method");
try
{
  DoWork();
}
catch (Exception e)
{
  AWSXRayRecorder.Instance.AddException(e);
}
finally
{
  AWSXRayRecorder.Instance.EndSubsegment();
}
```

當您在區段或其他子區段中建立子區段時，適用於 .NET 的 X-Ray 開發套件會為其產生 ID，並記錄開始時間和結束時間。

Example 使用中繼資料的子區段

```
"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"
```

```
    }
  },
```

使用適用於 .NET 的 X-Ray 開發套件將註釋和中繼資料新增至區段

您可以使用註釋和中繼資料記錄有關請求、環境或應用程式的其他資訊。您可以將註釋和中繼資料新增至 X-Ray SDK 建立的區段，或新增至您建立的自訂子區段。

註釋是具有字串、數字或布林值的鍵值對。註釋會編製索引，以便與篩選條件表達式搭配使用。使用標記記錄您想要用來在主控台將追蹤分組的資料，或是在呼叫 `GetTraceSummaries` API 時使用標記。

中繼資料是索引鍵/值對，可以具有任何類型的值，包括物件和清單，但不會編製索引以用於篩選條件表達式。使用中繼資料記錄您希望儲存在追蹤中的其他資料，但不需要搭配搜尋使用。

章節

- 使用適用於 .NET 的 X-Ray 開發套件記錄註釋
- 使用適用於 .NET 的 X-Ray 開發套件記錄中繼資料

使用適用於 .NET 的 X-Ray 開發套件記錄註釋

針對您想要建立索引以用於搜尋的區段或子區段，請使用標註來記錄這些區段上的資訊。

以下是 X-Ray 中所有註釋的必要項目：

註釋要求

- 金鑰 – X-Ray 註釋的金鑰最多可有 500 個英數字元。您不能使用點或句號以外的空格或符號 (。)
- 值 – X-Ray 註釋的值最多可有 1，000 個 Unicode 字元。
- 註釋數量 – 每個追蹤最多可以使用 50 個註釋。

記錄 AWS Lambda 函數外部的註釋

1. 取得 `AWSXRayRecorder` 的執行個體。

   ```
   using Amazon.XRay.Recorder.Core;
   ...
   AWSXRayRecorder recorder = AWSXRayRecorder.Instance;
   ```

2. 使用字串鍵、布林值、Int32、Int64、雙精確度浮點數或字串值，呼叫 `addAnnotation`。

```
recorder.AddAnnotation("mykey", "my value");
```

下列範例顯示如何使用包含點的putAnnotation字串索引鍵和布林值、數字或字串值來呼叫 。

```
document.putAnnotation("testkey.test", "my value");
```

記錄 AWS Lambda 函數內的註釋

Lambda 函數內的區段和子區段都由 Lambda 執行時間環境管理。如果您想要將註釋新增至 Lambda 函數內的區段或子區段，您必須執行下列動作：

1. 在 Lambda 函數內建立區段或子區段。
2. 將註釋新增至區段或子區段。
3. 結束區段或子區段。

下列程式碼範例示範如何將 註釋新增至 Lambda 函數內的子區段：

```
#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 會將註釋記錄為區段文件中annotations物件中的鍵值對。使用相同索引鍵呼叫addAnnotation操作兩次，會覆寫先前在相同區段或子區段上記錄的值。

若要尋找具有特定值註釋的追蹤，請在篩選條件表達式中使用 annotation[*key*]關鍵字。

使用適用於 .NET 的 X-Ray 開發套件記錄中繼資料

使用中繼資料來記錄區段或子區段的資訊，您不需要為在搜尋中使用編製索引。中繼資料值可以是字串、數字、布林值，或可序列化為 JSON 物件或陣列的任何其他物件。

記錄中繼資料

1. 取得 執行個體AWSXRayRecorder，如下列程式碼範例所示：

```
using Amazon.XRay.Recorder.Core;
...
AWSXRayRecorder recorder = AWSXRayRecorder.Instance;
```

2. AddMetadata 使用字串命名空間、字串索引鍵和物件值呼叫 ，如下列程式碼範例所示：

```
recorder.AddMetadata("my namespace", "my key", "my value");
```

   您也可以使用金鑰和值對來呼叫 AddMetadata操作，如下列程式碼範例所示：

```
recorder.AddMetadata("my key", "my value");
```

如果您未指定 命名空間的值，X-Ray SDK 會使用 default。使用相同索引鍵呼叫AddMetadata操作兩次，會覆寫先前在相同區段或子區段上記錄的值。

使用 Ruby

有兩種方式可以檢測您的 Ruby 應用程式，以將追蹤傳送至 X-Ray：

- AWS Distro for OpenTelemetry Ruby – 透過 AWS Distro for OpenTelemetry Collector， AWS 提供一組開放原始碼程式庫，用於將相關指標和追蹤傳送至多個 AWS 監控解決方案，包括 Amazon CloudWatch AWS X-Ray和 Amazon OpenSearch Service。
- AWS X-Ray 適用於 Ruby 的 SDK – 一組程式庫，用於透過 X-Ray 協助程式產生和傳送追蹤至 X-Ray。

如需詳細資訊，請參閱選擇 AWS Distro for OpenTelemetry 和 X-Ray SDKs。

AWS Distro for OpenTelemetry Ruby

使用 AWS Distro for OpenTelemetry (ADOT) Ruby，您可以檢測您的應用程式一次，並將相關指標和追蹤傳送至多個 AWS 監控解決方案 AWS X-Ray，包括 Amazon CloudWatch 和 Amazon OpenSearch Service。搭配 ADOT 使用 X-Ray 需要兩個元件：啟用 OpenTelemetry SDK 以搭配 X-Ray 使用，以及啟用 AWS Distro for OpenTelemetry Collector 以搭配 X-Ray 使用。

若要開始使用，請參閱 AWS Distro for OpenTelemetry Ruby 文件。

如需搭配 AWS AWS X-Ray 和其他 使用 Distro for OpenTelemetry 的詳細資訊 AWS 服務，請參閱 AWS Distro for OpenTelemetry 或 AWS Distro for OpenTelemetry 文件。

如需語言支援和用量的詳細資訊，請參閱 AWS GitHub 上的可觀測性。

AWS X-Ray 適用於 Ruby 的 SDK

X-Ray SDK 是 Ruby Web 應用程式的程式庫，提供產生追蹤資料並將其傳送至 X-Ray 協助程式的類別和方法。追蹤資料包含應用程式所提供服務的傳入 HTTP 請求，以及應用程式使用 AWS SDK、HTTP 用戶端或作用中記錄用戶端對下游服務進行的呼叫的相關資訊。您也可以手動建立區段，並將除錯資訊新增至註釋和中繼資料中。

您可以透過將它新增至您的 gemfile 並執行 `bundle install`，來下載軟體開發套件。

Example Gemfile

```
gem 'aws-sdk'
```

如果您使用 Rails，請先新增 X-Ray SDK 中介軟體來追蹤傳入的請求。請求篩選條件會建立區段。當區段開啟時，您可以使用軟體開發套件用戶端的方法，將資訊新增到區段，並建立子區段以追蹤下游呼叫。軟體開發套件也會在區段為開啟時自動記錄應用程式擲回的例外狀況。針對非 Rails 應用程式，您可以手動建立區段。

接著，使用 X-Ray 開發套件來檢測您的 適用於 Ruby 的 AWS SDK、HTTP 和 SQL 用戶端，方法是設定記錄器來修補相關聯的程式庫。每當您使用經檢測的用戶端呼叫下游 AWS 服務 或資源時，開發套件會在子區段中記錄有關呼叫的資訊。 AWS 服務 而您在服務中存取的資源會在追蹤地圖上顯示為下游節點，以協助您識別錯誤並調節個別連線的問題。

開始使用軟體開發套件之後，請設定記錄器以自訂其行為。您可以新增外掛程式，以記錄執行應用程式所需的運算資源相關資料、定義抽樣規則以自訂抽樣行為，並提供記錄器以在應用程式日誌中查看更多或更少的軟體開發套件資訊。

使用註釋與中繼資料，記錄應用程式所做的請求和工作等其他資訊。註釋是簡單的鍵/值對，系統會為其建立索引以用於篩選條件表達式，因此您可以搜尋包含特定資料的追蹤。中繼資料項目的限制性較低，可以記錄整個物件和陣列，任何可以序列化為 JSON 的項目。

> **標註與中繼資料**
>
> 註釋和中繼資料是您使用 X-Ray SDK 新增至區段的任意文字。註釋會編製索引，以便與篩選條件表達式搭配使用。中繼資料不會編製索引，但可以使用 X-Ray 主控台或 API 在原始區段中檢視。您授予 X-Ray 讀取存取權的任何人都可以檢視此資料。

當程式碼中有很多經過檢測的用戶端時，單一請求區段可能包含大量子區段，每個使用經檢測用戶端進行的呼叫都有一個子區段。您可以將用戶端呼叫包裝在自訂子區段中，以組織和群組子區段。您可以為整個函數或任何部分的程式碼建立自訂子區段，並記錄子區段上的中繼資料和註釋，而不必寫入父區段上的所有項目。

如需開發套件類別和方法的參考文件，請參閱AWS X-Ray 適用於 Ruby 的開發套件 API 參考。

要求

X-Ray SDK 需要 Ruby 2.3 或更新版本，並與下列程式庫相容：

- 適用於 Ruby 的 AWS SDK 3.0 版或更新版本
- Rails 版本 5.1 或更新版本

設定適用於 Ruby 的 X-Ray 開發套件

適用於 Ruby 的 X-Ray 開發套件具有名為 的類別`XRay.recorder`，可提供全域記錄器。您可以設定全域記錄器來自訂為傳入 HTTP 呼叫建立區段的中介軟體。

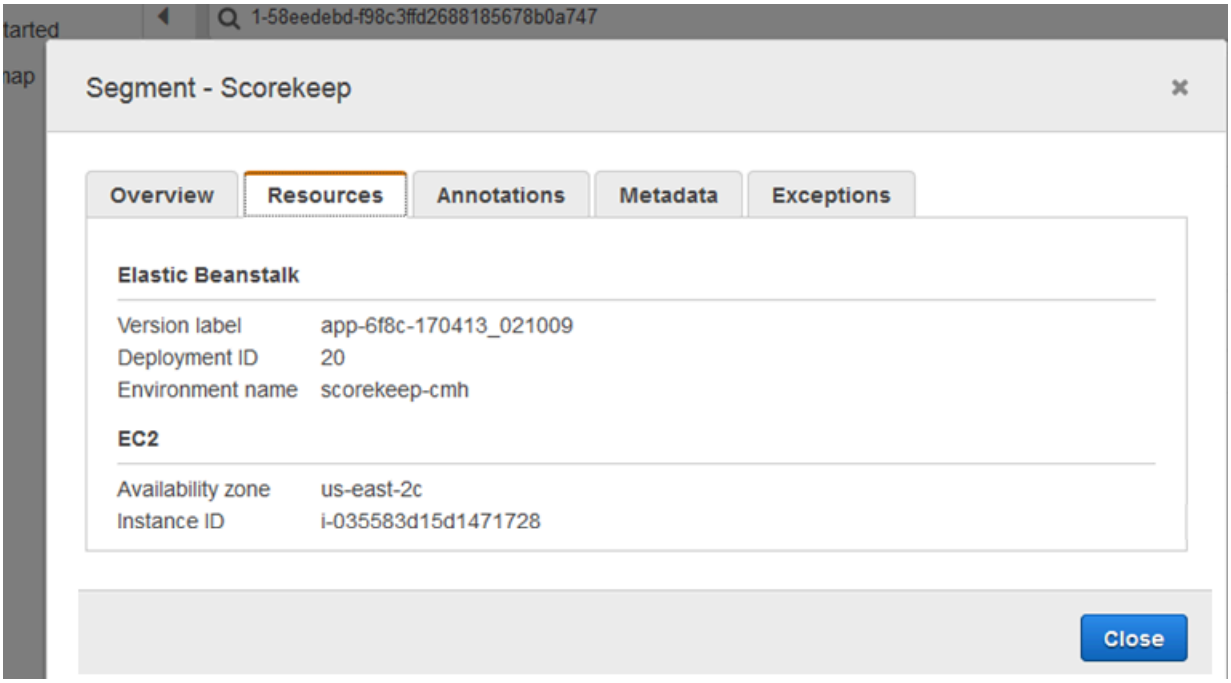
章節

服務外掛程式

使用 `plugins` 記錄託管您應用程式之服務的相關資訊。

外掛程式

- Amazon EC2 – `ec2` 新增執行個體 ID 和可用區域。
- Elastic Beanstalk – `elastic_beanstalk` 新增環境名稱、版本標籤和部署 ID。
- Amazon ECS – `ecs` 新增容器 ID。



若要使用外掛程式，請在您傳遞至記錄器的組態物件中指定它。

Example main.rb – 外掛程式組態

```
my_plugins = %I[ec2 elastic_beanstalk]

config = {
  plugins: my_plugins,
  name: 'my app',
}

XRay.recorder.configure(config)
```

您也可以使用優先於程式碼中設定值的環境變數，來設定記錄器。

軟體開發套件也會使用外掛程式設定來設定區段上的 `origin` 欄位。這表示執行您應用程式 AWS 的資源類型。當您使用多個外掛程式時，開發套件會使用下列解析順序來判斷原始伺服器：ElasticBeanstalk > EKS > ECS > EC2。

抽樣規則

SDK 會使用您在 X-Ray 主控台中定義的抽樣規則來決定要記錄哪些請求。預設規則每秒追蹤第一個請求，以及所有傳送追蹤至 X-Ray 服務的任何其他請求的 5%。在 X-Ray 主控台中建立其他規則，以自訂為每個應用程式記錄的資料量。

軟體開發套件會依定義自訂規則的順序進行套用。如果請求符合多個自訂規則，軟體開發套件只會套用第一個規則。

> **Note**
>
> 如果開發套件無法達到 X-Ray 以取得取樣規則，它會每秒還原為第一個請求的預設本機規則，以及每個主機任何額外請求的 5%。如果主機沒有呼叫取樣 APIs許可，或無法連線至 X-Ray 協助程式，而該常駐程式可做為 SDK 進行 API 呼叫的 TCP 代理，則可能會發生這種情況。

您也可以設定 SDK 以從 JSON 文件載入取樣規則。開發套件可以使用本機規則作為 X-Ray 取樣不可用的備份，或僅使用本機規則。

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
  }
}
```

此範例會定義一個自訂規則和預設規則。自訂規則會套用 5% 的取樣率，沒有追蹤 下路徑的最低請求數/api/move/。預設規則會追蹤每秒的第一個請求和 10% 的額外請求。

在本機定義規則的缺點是，固定目標是由記錄器的每個執行個體獨立套用，而不是由 X-Ray 服務管理。當您部署更多主機時，固定速率會乘以，使得難以控制記錄的資料量。

若要設定備份規則，請在您傳遞至記錄器的組態物件中定義文件的雜湊。

Example main.rb – 備份規則組態

```
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)
```

若要單獨存放抽樣規則，請在單獨的檔案中定義雜湊，然後要求 (require) 檔案來將其提取到您的應用程式中。

Example config/sampling-rules.rb

```
my_sampling_rules =  {
  version: 1,
  default: {
    fixed_target: 1,
    rate: 0.1
  }
}
```

Example main.rb – 從檔案取樣規則

```
require 'aws-xray-sdk'
require 'config/sampling-rules.rb'

config = {
  sampling_rules: my_sampling_rules,
  name: 'my app',
}
XRay.recorder.configure(config)
```

若要僅使用本機規則，請要求 (require) 抽樣規則並設定 `LocalSampler`。

Example main.rb – 本機規則取樣

```
require 'aws-xray-sdk'
require 'aws-xray-sdk/sampling/local/sampler'

config = {
  sampler: LocalSampler.new,
  name: 'my app',
}
XRay.recorder.configure(config)
```

您也可以設定全域記錄器來停用所有傳入請求的抽樣和檢測。

Example main.rb – 停用取樣

```
require 'aws-xray-sdk'
config = {
  sampling: false,
  name: 'my app',
}
XRay.recorder.configure(config)
```

日誌

根據預設，記錄器會將資訊層級的事件輸出到 `$stdout`。您可以透過在傳遞至記錄器的組態物件中定義 logger 來自定記錄日誌。

Example main.rb – 記錄

```
require 'aws-xray-sdk'
config = {
  logger: my_logger,
  name: 'my app',
}
XRay.recorder.configure(config)
```

當您手動產生子區段時，可使用除錯日誌來識別問題，例如未結束的子區段。

程式碼中的記錄器組態

可以從 `XRay.recorder` 上的 `configure` 方法取得其他可用設定。

- `context_missing` – 設定為 `LOG_ERROR`以避免在未開啟區段時，檢測程式碼嘗試記錄資料時擲出例外狀況。
- `daemon_address` – 設定 X-Ray 協助程式接聽程式的主機和連接埠。
- `name` – 設定 SDK 用於區段的服務名稱。
- `naming_pattern` – 設定網域名稱模式以使用動態命名。
- `plugins` – 使用外掛程式記錄應用程式 AWS 資源的相關資訊。
- `sampling` – 設定為 `false`以停用取樣。
- `sampling_rules` – 設定包含取樣規則的雜湊。

Example main.rb – 停用內容遺失例外狀況

```
require 'aws-xray-sdk'
config = {
  context_missing: 'LOG_ERROR'
}

XRay.recorder.configure(config)
```

使用 Rails 的記錄器組態

若您使用 Rails 框架，您可以在位於 `app_root/initializers` 之下的 Ruby 檔案中設定全域記錄器上的選項。X-Ray SDK 支援搭配 Rails 使用的額外組態金鑰。

- `active_record` – 設定為 `true`以記錄 Active Record 資料庫交易的子區段。

在名為 `Rails.application.config.xray` 的組態物件中設定可用設定。

Example config/initializers/aws_xray.rb

```
Rails.application.config.xray = {
  name: 'my app',
  patch: %I[net_http aws_sdk],
  active_record: true
}
```

環境變數

您可以使用環境變數來設定適用於 Ruby 的 X-Ray 開發套件。軟體開發套件支援以下變數：

- AWS_XRAY_TRACING_NAME – 設定 SDK 用於區段的服務名稱。覆寫您在 servlet 篩選條件的區段命名策略中設定的服務名稱。
- AWS_XRAY_DAEMON_ADDRESS – 設定 X-Ray 協助程式接聽程式的主機和連接埠。根據預設，開發套件會將追蹤資料傳送至 127.0.0.1:2000。如果您已設定協助程式在不同的連接埠上接聽，或在不同的主機上執行，請使用此變數。
- AWS_XRAY_CONTEXT_MISSING – 設定為 RUNTIME_ERROR 以在未開啟區段時，檢測程式碼嘗試記錄資料時擲回例外狀況。

  有效值

  - RUNTIME_ERROR – 捨棄執行時間例外狀況。
  - LOG_ERROR – 記錄錯誤並繼續 （ 預設 ）。
  - IGNORE_ERROR – 忽略錯誤並繼續。

  當您嘗試在未開啟請求時執行的啟動程式碼中，或在產生新執行緒的程式碼中，使用經檢測的用戶端時，可能會發生與缺少區段或子區段相關的錯誤。

環境變數會覆寫程式碼中所設的值。

使用適用於 Ruby 的 X-Ray 開發套件中介軟體追蹤傳入請求

您可以使用 X-Ray 開發套件來追蹤您的應用程式在 Amazon EC2EC2 AWS Elastic Beanstalk或 Amazon ECS 中的 EC2 執行個體上提供的傳入 HTTP 請求。

若您使用 Rails，請使用 Rails 中介軟體來檢測傳入的 HTTP 請求。當您將中介軟體新增至應用程式並設定區段名稱時，適用於 Ruby 的 X-Ray 開發套件會為每個抽樣請求建立區段。任何由額外檢測建立的區段都會成為子區段，位於提供 HTTP 請求及回應資訊的請求層級區段之下。此資訊包括時間、方法，以及請求的處置方式。

每個客群都有一個名稱，可在服務映射中識別您的應用程式。區段可以靜態命名，或者您可以設定 SDK，根據傳入請求中的主機標頭動態命名。動態命名可讓您根據請求中的網域名稱來分組追蹤，並在名稱不符合預期模式時套用預設名稱 （ 例如，如果主機標頭是偽造的 ）。

> **轉送的請求**
>
> 如果負載平衡器或其他中介裝置轉送請求到您的應用程式，X-Ray 會從請求中的 X-Forwarded-For標頭取得用戶端 IP，而不是從 IP 封包中的來源 IP 取得用戶端 IP。為轉送請求記錄的用戶端 IP 可能是偽造的，因此不應受信任。

轉送請求時，軟體開發套件會在區段中設定額外的欄位來指出這一點。如果區段包含`x_forwarded_for`設為 的欄位`true`，則用戶端 IP 會從 HTTP 請求中的 `X-Forwarded-For`標頭取得。

中介軟體會使用 `http` 區塊為每個傳入的請求建立區段，其中包含以下資訊：

- HTTP 方法 – GET、POST、PUT、DELETE 等。
- 用戶端地址 – 傳送請求之用戶端的 IP 地址。
- 回應碼 – 已完成請求的 HTTP 回應碼。
- 時間 – 開始時間（收到請求的時間）和結束時間（傳送回應的時間）。
- 使用者代理程式 — 來自請求`user-agent`的 。
- 內容長度 — `content-length`來自回應的 。

使用 Rails 中介軟體

若要使用中介軟體，請更新您的 gemfile 以包含必要的 railtie。

Example Gemfile - rails

```
gem 'aws-xray-sdk', require: ['aws-xray-sdk/facets/rails/railtie']
```

若要使用中介軟體，您還必須使用代表追蹤映射中應用程式的名稱來設定記錄器。

Example config/initializers/aws_xray.rb

```
Rails.application.config.xray = {
  name: 'my app'
}
```

手動檢測程式碼

若您沒有使用 Rails，請手動建立區段。您可以為每個傳入請求建立區段，或在修補的 HTTP 或 AWS SDK 用戶端周圍建立區段，以提供記錄器新增子區段的內容。

```
# 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
```

設定區段命名策略

AWS X-Ray 使用服務名稱來識別您的應用程式，並將其與應用程式使用的其他應用程式、資料庫、外部 APIs AWS 和資源區分開來。當 X-Ray SDK 為傳入請求產生區段時，它會在區段的名稱欄位中記錄應用程式的服務名稱。

X-Ray SDK 可以在 HTTP 請求標頭中的主機名稱之後命名區段。不過，此標頭可能是偽造的，這可能會導致服務映射中出現非預期的節點。若要防止 SDK 因具有偽造主機標頭的請求而不正確命名區段，您必須指定傳入請求的預設名稱。

如果您的應用程式為多個網域提供請求，您可以將 SDK 設定為使用動態命名策略，以在區段名稱中反映這一點。動態命名策略允許 SDK 將主機名稱用於符合預期模式的請求，並將預設名稱套用至不符合的請求。

例如，您可能有一個應用程式向三個子網域提供請求：`www.example.com`、 `api.example.com`和 `static.example.com`。您可以使用動態命名策略搭配 模式`*.example.com`，以不同名稱識別每個子網域的區段，導致服務映射上有三個服務節點。如果您的應用程式收到主機名稱不符合模式的請求，您會在服務映射上看到具有您指定之備用名稱的第四個節點。

若要為所有請求區段使用相同的名稱，請如上一節所述，在設定記錄器時指定您應用程式的名稱。

動態命名策略可定義主機名稱應相符的模式，以及如果 HTTP 請求中的主機名稱不符合模式時要使用的預設名稱。若要動態命名區段，請在組態雜湊中指定命名模式。

Example main.rb – 動態命名

```
config = {
  naming_pattern: '*mydomain*',
  name: 'my app',
}

XRay.recorder.configure(config)
```

您可以在模式中使用 '*' 來比對任何字串，或是 '?' 來比對任何單一字元。

> **Note**
>
> 您可以使用 AWS_XRAY_TRACING_NAME 環境變數來覆寫您在程式碼中定義的預設服務名稱。

修補程式庫來檢測下游呼叫

若要檢測下游呼叫，請使用適用於 Ruby 的 X-Ray 開發套件來修補應用程式使用的程式庫。適用於 Ruby 的 X-Ray 開發套件可以修補下列程式庫。

支援的程式庫

- net/http – 檢測 HTTP 用戶端。
- aws-sdk – 檢測 適用於 Ruby 的 AWS SDK 用戶端。

當您使用修補的程式庫時，適用於 Ruby 的 X-Ray 開發套件會為呼叫建立子區段，並記錄請求和回應中的資訊。區段必須透過軟體開發套件中介軟體或呼叫 XRay.recorder.begin_segment 供軟體開發套件使用，以建立子區段。

若要修補程式庫，請在您傳遞給 X-Ray 記錄器的組態物件中指定它們。

Example main.rb – 修補程式程式庫

```
require 'aws-xray-sdk'

config = {
  name: 'my app',
```

```
  patch: %I[net_http aws_sdk]
}

XRay.recorder.configure(config)
```

使用適用於 Ruby 的 X-Ray AWS 開發套件追蹤 SDK 呼叫

當您的應用程式呼叫 AWS 服務 來存放資料、寫入佇列或傳送通知時，適用於 Ruby 的 X-Ray 開發套件會在子區段中追蹤下游的呼叫。您在這些服務中存取的追蹤 AWS 服務 和資源 （例如 Amazon S3 儲存貯體或 Amazon SQS 佇列），會在 X-Ray 主控台的追蹤地圖上顯示為下游節點。

當您修補程式aws-sdk庫時，適用於 Ruby 的 X-Ray 開發套件會自動檢測所有 AWS SDK 用戶端。您無法檢測個別用戶端。

對於所有 服務，您可以在 X-Ray 主控台中查看名為 的 API 名稱。對於服務子集，X-Ray SDK 會將資訊新增至區段，以在服務映射中提供更精細的 。

例如，當您使用經檢測的 DynamoDB 用戶端進行呼叫時，軟體開發套件會將資料表名稱新增至以資料表為目標的呼叫區段。在 主控台中，每個資料表都會在服務映射中顯示為個別節點，並具有一般 DynamoDB 節點，用於非以資料表為目標的呼叫。

Example 呼叫 DynamoDB 以儲存項目的子區段

```
{
  "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",
  }
}
```

您存取具名資源時，對以下服務的呼叫會在服務地圖中建立額外節點。未針對特定資源的呼叫，則會建立服務的一般節點。

- Amazon DynamoDB – 資料表名稱
- Amazon Simple Storage Service – 儲存貯體和金鑰名稱
- Amazon Simple Queue Service – 佇列名稱

使用 X-Ray SDK 產生自訂子區段

子區段會延伸追蹤的區段，其中包含為處理請求而完成之工作的詳細資訊。每次您與經檢測的用戶端進行呼叫時，X-Ray 開發套件都會記錄子區段中產生的資訊。您可以建立其他子區段來將其他子區段分組、測量程式碼區段的效能，或記錄註釋和中繼資料。

若要管理子區段，請使用 `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
```

若要建立函數的子區段，請將它包裝在對 `XRay.recorder.capture` 進行的呼叫中。

```
XRay.recorder.capture('name_for_subsegment') do |subsegment|
  resp = myfunc() # myfunc is your function
  subsegment.annotations.update k1: 'v1'
  resp
end
```

當您在區段或其他子區段中建立子區段時，X-Ray SDK 會為其產生 ID，並記錄開始時間和結束時間。

Example 使用中繼資料的子區段

```
"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"
```

```
    }
  },
```

使用適用於 Ruby 的 X-Ray 開發套件將註釋和中繼資料新增至區段

您可以使用註釋和中繼資料記錄有關請求、環境或應用程式的其他資訊。您可以將註釋和中繼資料新增至 X-Ray SDK 建立的區段，或新增至您建立的自訂子區段。

註釋是具有字串、數字或布林值的鍵值對。註釋會編製索引，以便與篩選條件表達式搭配使用。使用標記記錄您想要用來在主控台將追蹤分組的資料，或是在呼叫 `GetTraceSummaries` API 時使用標記。

中繼資料是索引鍵/值對，可以具有任何類型的值，包括物件和清單，但不會編製索引以用於篩選條件表達式。使用中繼資料記錄您希望儲存在追蹤中的其他資料，但不需要搭配搜尋使用。

除了註釋和中繼資料，您也可以在區段上記錄使用者 ID 字串。區段會將使用者 ID 記錄在單獨的欄位中，並建立索引以用於搜尋。

章節

- 使用適用於 Ruby 的 X-Ray 開發套件記錄註釋
- 使用適用於 Ruby 的 X-Ray 開發套件記錄中繼資料
- 使用適用於 Ruby 的 X-Ray 開發套件記錄使用者 IDs

使用適用於 Ruby 的 X-Ray 開發套件記錄註釋

針對您想要建立索引以用於搜尋的區段或子區段，請使用標註來記錄這些區段上的資訊。

註釋要求

- 金鑰 – X-Ray 註釋的金鑰最多可有 500 個英數字元。您不能使用點或句號以外的空格或符號 (。)
- 值 – X-Ray 註釋的值最多可有 1，000 個 Unicode 字元。
- 註釋數量 – 每個追蹤最多可以使用 50 個註釋。

記錄標註

1. 從 `xray_recorder` 取得目前區段或子區段的參考。

```
require 'aws-xray-sdk'
...
```

```
document = XRay.recorder.current_segment
```

或

```
require 'aws-xray-sdk'
...
document = XRay.recorder.current_subsegment
```

2. 使用雜湊值呼叫 update。

```
my_annotations = { id: 12345 }
document.annotations.update my_annotations
```

以下是示範如何使用包含點的update註釋索引鍵呼叫 的範例。

```
my_annotations = { testkey.test: 12345 }
document.annotations.update my_annotations
```

軟體開發套件會將標註以鍵/值對記錄在區段文件中的 annotations 物件內。若使用相同鍵呼叫 add_annotations 兩次，則會覆寫之前在相同區段或子區段上記錄的值。

若要尋找具有特定值註釋的追蹤，請在篩選條件表達式中使用 annotation[*key*]關鍵字。

使用適用於 Ruby 的 X-Ray 開發套件記錄中繼資料

針對您不想要建立索引以用於搜尋的區段，請使用中繼資料來記錄這些區段或子區段上的資訊。中繼資料值可以是字串、數字、布林值，或可序列化為 JSON 物件或陣列的任何物件。

記錄中繼資料

1. 從 xray_recorder 取得目前區段或子區段的參考。

```
require 'aws-xray-sdk'
...
document = XRay.recorder.current_segment
```

或

```
require 'aws-xray-sdk'
...
```

```
document = XRay.recorder.current_subsegment
```

2. 使用字串鍵、布林值、數字、字串或物件值，以及字串命名空間，呼叫 metadata。

```
my_metadata = {
  my_namespace: {
    key: 'value'
  }
}
subsegment.metadata my_metadata
```

若使用相同鍵呼叫 metadata 兩次，則會覆寫之前在相同區段或子區段上記錄的值。

使用適用於 Ruby 的 X-Ray 開發套件記錄使用者 IDs

記錄請求區段上的使用者 ID 以識別傳送請求的使用者。

記錄使用者 ID

1. 從 xray_recorder 取得目前區段的參考。

```
require 'aws-xray-sdk'
...
document = XRay.recorder.current_segment
```

2. 將區段上的使用者欄位設為傳送請求的使用者字串 ID。

```
segment.user = 'U12345'
```

您可以在控制器中設定使用者，以在應用程式開始處理請求時馬上記錄使用者 ID。

若要尋找使用者 ID 的追蹤，請在篩選條件表達式中使用 user關鍵字。

從 X-Ray 檢測遷移至 OpenTelemetry 檢測

X-Ray 正在轉換為 OpenTelemetry (OTel)，作為應用程式追蹤和可觀測性的主要檢測標準。此策略轉移 AWS 符合業界最佳實務，並為客戶提供更全面、靈活且面向未來的可觀測性需求解決方案。OpenTelemetry 在業界的廣泛採用可讓跨不同系統追蹤請求，包括可能未直接與 X-Ray 整合 AWS 的外部請求。

本章提供順暢轉換的建議，並強調遷移至 OpenTelemetry 型解決方案的重要性，以確保持續支援和存取應用程式檢測和可觀測性的最新功能。

建議採用 OpenTelemetry 作為檢測應用程式的可觀測性解決方案。

主題

了解 OpenTelemetry

OpenTelemetry 是一種業界標準的可觀測性架構，可提供標準化通訊協定和工具來收集遙測資料。它提供統一方法來檢測、產生、收集和匯出遙測資料，例如指標、日誌和追蹤。

當您從 X-Ray SDKs遷移至 OpenTelemetry 時，可獲得下列優點：

- 增強架構和程式庫檢測支援
- 支援其他程式設計語言
- 自動檢測功能
- 彈性抽樣組態選項

- 指標、日誌和追蹤的統一集合

OpenTelemetry 收集器提供比 X-Ray 協助程式更多的資料收集格式和匯出目的地選項。

中的 OpenTelemetry 支援 AWS

AWS 提供多種使用 OpenTelemetry 的解決方案：

- AWS Distro for OpenTelemetry

  將 OpenTelemetry 追蹤作為區段匯出至 X-Ray。

  如需詳細資訊，請參閱 AWS Distro for OpenTelemetry。
- CloudWatch Application Signals

  匯出自訂 OpenTelemetry 追蹤和指標以監控應用程式運作狀態。

  如需詳細資訊，請參閱使用 Application Signals。
- CloudWatch OTel 端點

  使用 HTTP OTel 端點搭配原生 OpenTelemetry 檢測，將 OpenTelemetry 追蹤匯出至 X-Ray。

  如需詳細資訊，請參閱使用 OTel 端點。

搭配 AWS CloudWatch 使用 OpenTelemetry

AWS CloudWatch 透過用戶端應用程式檢測和 native AWS CloudWatch 服務支援 OpenTelemetry 追蹤，例如 Application Signals、Trace、Map、Metrics 和 Logs。如需詳細資訊，請參閱 OpenTelemetry。

了解遷移的 OpenTelemetry 概念

下表將 X-Ray 概念映射至其 OpenTelemetry 對等項目。了解這些映射可協助您將現有的 X-Ray 檢測轉換為 OpenTelemetry：

X-Ray 概念	OpenTelemetry 概念
X-Ray 記錄器	追蹤器供應商和追蹤器

X-Ray 概念	OpenTelemetry 概念
服務外掛程式	資源偵測器
區段	（伺服器）跨度
子區段	（非伺服器）跨度
X-Ray 取樣規則	OpenTelemetry 取樣（可自訂）
X-Ray 發射器	跨度匯出工具（可自訂）
註釋/中繼資料	Attributes
程式庫檢測	程式庫檢測
X-Ray 追蹤內容	跨度內容
X-Ray 追蹤內容傳播	W3C 追蹤內容傳播
X-Ray 追蹤取樣	OpenTelemetry 追蹤取樣
N/A	跨度處理
N/A	包裝
X-Ray 協助程式	OpenTelemetry Collector

> **Note**
>
> 如需 OpenTelemetry 概念的詳細資訊，請參閱 OpenTelemetry 文件。

比較功能

下表顯示兩個 服務都支援哪些功能。使用此資訊來識別您在遷移期間需要解決的任何差距：

功能	X-Ray 檢測	OpenTelemetry 檢測
程式庫檢測	支援	支援
X-Ray 取樣	支援	OTel Java/ 支援。NET/Go ADOT Java/ 支援。NET/Python/Node.js
X-Ray 追蹤內容傳播	支援	支援
資源偵測	支援	支援
區段註釋	支援	支援
區段中繼資料	支援	支援
零碼自動檢測	Java 支援	OTel Java/ 支援。NET/Python/Node.js ADOT Java/ 支援。NET/Python/Node.js
手動追蹤建立	支援	支援

設定和設定追蹤

若要在 OpenTelemetry 中建立追蹤，您需要一個追蹤器。您可以透過在應用程式中初始化追蹤器提供者來取得追蹤器。這類似於您使用 X-Ray 記錄器來設定 X-Ray，以及在 X-Ray 追蹤中建立區段和子區段的方式。

> **Note**
>
> OpenTelemetry Tracer 提供者提供的組態選項比 X-Ray 記錄器更多。

了解追蹤資料結構

了解基本概念和功能映射後，您可以了解特定實作詳細資訊，例如追蹤資料結構和取樣。

OpenTelemetry 使用跨度而非區段和子區段來建構追蹤資料。每個範圍都包含下列元件：

- 名稱
- 唯一 ID
- 開始和結束時間戳記
- 跨度類型
- 範圍內容
- 屬性（鍵值中繼資料）
- 事件（時間戳記日誌）
- 其他範圍的連結
- 狀態資訊
- 父系跨度參考

當您遷移至 OpenTelemetry 時，您的範圍會自動轉換為 X-Ray 區段或子區段。這可確保您現有的 CloudWatch 主控台體驗保持不變。

使用跨度屬性

X-Ray 開發套件提供兩種將資料新增至區段和子區段的方式：

註釋

索引用於篩選和搜尋的鍵值對

中繼資料

索引鍵/值組包含未編製索引以進行搜尋的複雜資料

根據預設，OpenTelemetry 跨度屬性會轉換為 X-Ray 原始資料中的中繼資料。若要改為將特定屬性轉換為註釋，請將其索引鍵新增至`aws.xray.annotations`屬性清單。

- 如需 OpenTelemetry 概念的詳細資訊，請參閱 OpenTelemetry 追蹤
- 如需 OpenTelemetry 資料如何映射到 X-Ray 資料的詳細資訊，請參閱 OpenTelemetry 到 X-Ray 資料模型轉譯

偵測您環境中的資源

OpenTelemetry 使用 Resource Detectors 收集有關產生遙測資料之資源的中繼資料。此中繼資料會儲存為資源屬性。例如，產生遙測的實體可以是 Amazon ECS 叢集或 Amazon EC2 執行個體，而且可以從這些實體記錄的資源屬性可以包含 Amazon ECS 叢集 ARN 或 Amazon EC2 執行個體 ID。

- 如需支援的資源類型相關資訊，請參閱 OpenTelemetry Resource 語意慣例
- 如需有關 X-Ray 服務外掛程式的資訊，請參閱設定 X-Ray SDK

管理抽樣策略

追蹤抽樣透過從代表性的請求子集而非所有請求收集資料，來協助您管理成本。OpenTelemetry 和 X-Ray 都支援取樣，但實作方式不同。

> **Note**
>
> 取樣少於 100% 的追蹤可減少可觀測性成本，同時保持對應用程式效能的有意義洞察。

OpenTelemetry 提供數種內建的取樣策略，並可讓您建立自訂的取樣策略。您也可以使用某些 SDK 語言設定 X-Ray 遠端取樣器，以搭配 OpenTelemetry 使用 X-Ray 取樣規則。

OpenTelemetry 的其他抽樣策略包括：

- 父系抽樣 – 在套用其他抽樣策略之前，遵守父系的抽樣決策
- 追蹤 ID 比率型取樣 – >隨機取樣指定的跨度百分比
- 尾端取樣 – 套用取樣規則以完成 OpenTelemetry Collector 中的追蹤
- 自訂取樣器 – 使用取樣界面實作您自己的取樣邏輯

如需有關 X-Ray 取樣規則的資訊，請參閱 X-Ray 主控台中的取樣規則

如需 OpenTelemetry 尾部取樣的相關資訊，請參閱尾部取樣處理器

管理追蹤內容

X-Ray SDKs會管理區段內容，以正確地處理追蹤中區段和子區段之間的父子關係。OpenTelemetry 使用類似的機制來確保跨度具有正確的父系跨度。它會在整個請求內容中存放和傳播追蹤資料。例如，

當您的應用程式處理請求並建立代表該請求的伺服器範圍時，OpenTelemetry 會將伺服器範圍存放在 OpenTelemetry 內容中，以便在建立子範圍時，該子範圍可以參考內容中的範圍作為其父系。

傳播追蹤內容

X-Ray 和 OpenTelemetry 都使用 HTTP 標頭跨服務傳播追蹤內容。這可讓您連結不同 服務產生的追蹤資料，並維護抽樣決策。

X-Ray SDK 會使用 X-Ray 追蹤標頭自動傳播追蹤內容。當一個服務呼叫另一個服務時，追蹤標頭包含維持追蹤之間父子關係所需的內容。

OpenTelemetry 支援多個追蹤標頭格式進行內容傳播，包括：

- W3C 追蹤內容 （預設）
- X-Ray 追蹤標頭
- 其他自訂格式

> **Note**
>
> 您可以設定 OpenTelemetry 使用一或多個標頭格式。例如，使用 X-Ray 傳播器將追蹤內容傳送至支援 X-Ray 追蹤 AWS 的服務。

設定並使用 X-Ray 傳播器來啟用跨 AWS 服務的追蹤。這可讓您將追蹤內容傳播至 API Gateway 端點和其他支援 X-Ray 的服務。

- 如需有關 X-Ray 追蹤標頭的資訊，請參閱《X-Ray 開發人員指南》中的追蹤標頭
- 如需有關 OpenTelemetry 內容傳播的資訊，請參閱 OpenTelemetry 文件中的內容和內容傳播

使用程式庫檢測

X-Ray 和 OpenTelemetry 都提供程式庫檢測，需要最少的程式碼變更，才能將追蹤新增至您的應用程式。

X-Ray 提供程式庫檢測功能。這可讓您以最少的應用程式程式碼變更來新增預先建置的 X-Ray 檢測。這些檢測支援 AWS SDK 和 HTTP 用戶端等特定程式庫，以及 Spring Boot 或 Express.js 等 Web 架構。

OpenTelemetry 的檢測程式庫會透過程式庫掛鉤或自動程式碼修改，為您的程式庫產生詳細的跨度，因此需要最少的程式碼變更。

若要判斷 OpenTelemetry 的程式庫檢測是否支援您的程式庫，請在 OpenTelemetry 登錄檔中的 OpenTelemetry 登錄檔中搜尋它。

匯出追蹤

X-Ray 和 OpenTelemetry 使用不同的方法來匯出追蹤資料。

X-Ray 追蹤匯出

X-Ray SDKs使用發射器來傳送追蹤資料：

- 將區段和子區段傳送至 X-Ray 協助程式
- 將 UDP 用於非封鎖 I/O
- 依預設在 SDK 中設定

OpenTelemetry 追蹤匯出

OpenTelemetry 使用可設定的跨度匯出工具來傳送追蹤資料：

- 使用 http/protobuf 或 grpc 通訊協定
- 匯出範圍至 OpenTelemetry Collector 或 CloudWatch Agent 監控的端點
- 允許自訂匯出工具組態

處理和轉送追蹤

X-Ray 和 OpenTelemetry 都提供元件來接收、處理和轉送追蹤資料。

X-Ray 追蹤處理

X-Ray 協助程式處理追蹤處理：

- 從 X-Ray SDKs接聽 UDP 流量
- 批次區段和子區段
- 將批次上傳至 X-Ray 服務

OpenTelemetry 追蹤處理

OpenTelemetry Collector 會處理追蹤處理：

- 接收來自檢測服務的追蹤
- 處理和選擇性修改追蹤資料
- 將已處理的追蹤傳送至各種後端，包括 X-Ray

> **Note**
>
> AWS CloudWatch Agent 也可以接收 OpenTelemetry 追蹤並將其傳送至 X-Ray。如需詳細資訊，請參閱使用 OpenTelemetry 收集指標和追蹤。

跨度處理 (OpenTelemetry 特定概念)

OpenTelemetry 使用跨處理器修改建立的跨度：

- 允許在建立或完成時讀取和修改範圍
- 啟用跨度處理的自訂邏輯

套件 (OpenTelemetry-soecific 概念)

OpenTelemetry 的套件功能允許傳播鍵值資料：

- 啟用將任意資料與追蹤內容一起傳遞
- 適用於跨服務界限傳播應用程式特定資訊

如需 OpenTelemetry Collector 的詳細資訊，請參閱 OpenTelemetry Collector

如需有關 X-Ray 概念的資訊，請參閱《X-Ray 開發人員指南》中的 X-Ray 概念

遷移概觀

本節提供遷移所需的程式碼變更概觀。以下列出特定語言指引和 X-Ray 協助程式遷移步驟。

> **Important**
>
> 若要從 X-Ray 檢測完全遷移到 OpenTelemetry 檢測，您需要：
>
> 1. 使用 OpenTelemetry 解決方案取代 X-Ray SDK 用量
> 2. 將 X-Ray 協助程式取代為 CloudWatch Agent 或 OpenTelemetry Collector （使用 X-Ray Exporter)

- 遷移至 OpenTelemetry Java
- 遷移至 OpenTelemetry Go
- 遷移至 OpenTelemetry Node.js
- 遷移至 OpenTelemetry .NET
- 遷移至 OpenTelemetry Python
- 遷移至 OpenTelemetry Ruby

新應用程式和現有應用程式的建議

對於新的和現有的應用程式，建議使用下列解決方案在您的應用程式中啟用追蹤：

檢測

- OpenTelemetry SDKs
- AWS Distro for OpenTelemetry 檢測

資料收集

- OpenTelemetry Collector
- CloudWatch 代理程式

遷移至 OpenTelemetry 型解決方案後，您的 CloudWatch 體驗將保持不變。您仍然可以在 CloudWatch 主控台的追蹤和追蹤地圖頁面中檢視相同格式的追蹤，或透過 X-Ray APIs擷取追蹤資料。

追蹤設定變更

您需要將 X-Ray 設定取代為 OpenTelemetry 設定。

X-Ray 和 OpenTelemetry 設定的比較

功能	X-Ray 開發套件	OpenTelemetry
預設組態	• X-Ray 集中式取樣 • X-Ray 追蹤內容傳播 • 追蹤匯出至 X-Ray 協助程式	• 將追蹤匯出至 OpenTelemetry Collector 或 CloudWatch Agent (HTTP/gRPC) • W3C 追蹤內容傳播
手動組態	• 本機取樣規則 • 資源偵測外掛程式	• X-Ray 取樣（可能無法用於所有語言） • 資源偵測 • X-Ray 追蹤內容傳播

程式庫檢測變更

更新您的程式碼以使用 OpenTelemetry Library Instrumentation，而非適用於 AWS SDK、HTTP 用戶端、Web Framework 和其他程式庫的 X-Ray Library Instrumentation。這會產生 OpenTelemetry Traces，而不是 X-Ray Traces。

> **Note**
>
> 程式碼變更會因語言和程式庫而異。如需詳細說明，請參閱語言特定的遷移指南。

Lambda 環境檢測變更

若要在 Lambda 函數中使用 OpenTelemetry，請選擇下列其中一個設定選項：

1. 使用自動檢測 Lambda Layer：
   - （建議）CloudWatch Application Signals Lambda layer

     > **Note**
     >
     > 若要僅使用追蹤，請設定 Lambda 環境變數 `OTEL_AWS_APPLICATION_SIGNALS_ENABLED=false`。

   - AWS ADOT 的 受管 Lambda Layer

2. 手動設定 Lambda 函數的 OpenTelemetry：
   - 使用 X-Ray UDP 範圍匯出器設定簡易範圍處理器
   - 設定 X-Ray Lambda 傳播器

手動建立追蹤資料

將 X-Ray 區段和子區段取代為 OpenTelemetry 跨度：

- 使用 OpenTelemetry Tracer 建立跨度
- 新增屬性至 Spans（相當於 X-Ray 中繼資料和註釋）

> **Important**
>
> 傳送至 X-Ray 時：
>
> - 伺服器跨度轉換為 X-Ray 區段
> - 其他跨度轉換為 X-Ray 子區段
> - 屬性預設會轉換為中繼資料

若要將屬性轉換為註釋，請將其索引鍵新增至`aws.xray.annotations`屬性清單。如需詳細資訊，請參閱啟用自訂 X-Ray 註釋。

從 X-Ray Daemon 遷移至 AWS CloudWatch 代理程式或 OpenTelemetry 收集器

您可以使用 CloudWatch 代理程式或 OpenTelemetry 收集器，從經檢測的應用程式接收追蹤，並將其傳送至 X-Ray。

> **Note**
>
> CloudWatch 代理程式 1.300025.0 版和更新版本可以收集 OpenTelemetry 追蹤。使用 CloudWatch 代理程式而非 X-Ray 協助程式可減少您需要管理的代理程式數量。如需詳細資訊，請參閱使用 CloudWatch 代理程式收集指標、日誌和追蹤。

章節

在 Amazon EC2 或內部部署伺服器上遷移

> **Important**
>
> 先停止 X-Ray Daemon 程序，再使用 CloudWatch 代理程式或 OpenTelemetry 收集器來防止連接埠衝突。

現有的 X-Ray 協助程式設定

安裝協助程式

您使用下列其中一種方法來安裝現有的 X-Ray 協助程式用量：

手動安裝

從 X-Ray 協助程式 Amazon S3 儲存貯體下載並執行可執行檔。

自動安裝

啟動執行個體時，請使用此指令碼來安裝協助程式：

```
#!/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
```

設定 精靈

您現有的 X-Ray Daemon 用量已使用下列其中一種方式設定：

- 命令列引數
- 組態檔案 (`xray-daemon.yaml`)

Example 使用組態檔

```
./xray -c ~/xray-daemon.yaml
```

執行精靈

您現有的 X-Ray Daemon 用量已使用下列命令啟動：

```
~/xray-daemon$ ./xray -o -n us-east-1
```

移除協助程式

若要從 Amazon EC2 執行個體中移除 X-Ray 協助程式：

1. 停止協助程式服務：

   ```
   systemctl stop xray
   ```

2. 刪除組態檔案：

   ```
   rm ~/path/to/xray-daemon.yaml
   ```

3. 如果已設定，請移除日誌檔案：

   > **Note**
   >
   > 日誌檔案位置取決於您的組態：
   >
   > - 命令列組態：`/var/log/xray-daemon.log`
   > - 組態檔案：檢查`LogPath`設定

設定 CloudWatch 代理程式

安裝代理程式

如需安裝說明，請參閱在內部部署伺服器上安裝 CloudWatch 代理程式。

設定代理程式

1. 建立組態檔案以啟用追蹤集合。如需詳細資訊，請參閱建立 CloudWatch 代理程式組態檔案。

2. 設定 IAM 許可：
   - 連接 IAM 角色或指定代理程式的登入資料。如需詳細資訊，請參閱設定 IAM 角色。
   - 請確定角色或登入資料包含 `xray:PutTraceSegments`許可。

啟動 代理程式

如需啟動代理程式的指示，請參閱使用命令列啟動 CloudWatch 代理程式。

設定 OpenTelemetry 收集器

安裝收集器

為您的作業系統下載並安裝 OpenTelemetry 收集器。如需說明，請參閱安裝收集器。

設定收集器

在收集器中設定下列元件：

- awsproxy 延伸模組

  X-Ray 取樣的必要項目
- OTel 接收器

  從您的應用程式收集追蹤
- xray 匯出工具

  將追蹤傳送至 X-Ray

Example 範例收集器組態 — 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`許可設定 AWS 登入資料。如需詳細資訊，請參閱指定登入資料。

啟動收集器

使用您的組態檔案執行收集器：

```
otelcol --config=otel-collector-config.yaml
```

在 Amazon ECS 上遷移

> **Important**
>
> 您的任務角色必須擁有您使用的任何收集器的 `xray:PutTraceSegments` 許可。
> 在相同主機上執行 CloudWatch 代理程式或 OpenTelemetry 收集器容器之前，請先停止任何現有的 X-Ray 協助程式容器，以防止連接埠衝突。

使用 CloudWatch 代理程式

1. 從 Amazon ECR Public Gallery 取得 Docker 映像。

2. 建立名為 的組態檔案cw-agent-otel.json：

```
{
  "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. 將組態存放在 Systems Manager 參數存放區：

   1. 開啟 https：//https://console.aws.amazon.com/systems-manager/
   2. 選擇建立參數
   3. 輸入下列值：
      - 名稱：/ecs/cwagent/otel-config
      - 層：標準
      - 類型：字串
      - 資料類型：文字
      - 值：【在此貼上 cw-agent-otel.json 組態】

4. 使用橋接網路模式建立任務定義：

   在您的任務定義中，組態取決於您使用的聯網模式。預設值是橋接聯網，可在您的預設 VPC 中使用。在橋接網路中，設定 OTEL_EXPORTER_OTLP_TRACES_ENDPOINT環境變數，以告知 OpenTelemetry SDK CloudWatch 代理程式的端點和連接埠。您也應該建立從應用程式容器到收集器容器的連結，以便從應用程式中的 OpenTelemetry SDK 將追蹤傳送至收集器容器。

   Example CloudWatch 代理程式任務定義

```
{
    "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"
                }
            ]
        }
    ]
}
```

如需詳細資訊，請參閱部署 CloudWatch 代理程式以在 Amazon ECS 上收集 Amazon EC2 執行個體層級指標。

使用 OpenTelemetry 收集器

1. `otel/opentelemetry-collector-contrib` 從 Docker Hub 取得 Docker 映像。
2. `otel-collector-config.yaml` 使用 Amazon EC2 設定收集器區段中所示的相同內容建立名為的組態檔案，但更新端點以使用 `0.0.0.0`而非 `127.0.0.1`。
3. 若要在 Amazon ECS 中使用此組態，您可以將組態存放在 Systems Manager 參數存放區中。首先，前往 Systems Manager 參數存放區主控台，然後選擇建立新參數 。使用下列資訊建立新的參數：
   - 名稱：/ecs/otel/config （此名稱將在收集器的任務定義中參考）
   - 層：標準
   - 類型：字串
   - 資料類型：文字
   - 值：【在此貼上 otel-collector-config.yaml 組態】
4. 建立任務定義，以使用橋接網路模式做為範例來部署 OpenTelemetry 收集器。

   在任務定義中，組態取決於您使用的聯網模式。預設值是橋接聯網，可在您的預設 VPC 中使用。在橋接網路中，設定 `OTEL_EXPORTER_OTLP_TRACES_ENDPOINT`環境變數，以告知 OpenTelemetry SDK OpenTelemetry Collector 的端點和連接埠。您也應該建立從應用程式容器到收集器容器的連結，以便從應用程式中的 OpenTelemetry SDK 將追蹤傳送至收集器容器。

   Example OpenTelemetry 收集器任務定義

```
{
    "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 上遷移

> **Important**
>
> 先停止 X-Ray Daemon 程序，再使用 CloudWatch 代理程式來防止連接埠衝突。

您現有的 X-Ray 協助程式整合已使用 Elastic Beanstalk 主控台開啟，或使用組態檔案在應用程式原始程式碼中設定 X-Ray 協助程式。

使用 CloudWatch 代理程式

在 Amazon Linux 2 平台上，使用.ebextensions組態檔案設定 CloudWatch 代理程式：

1. 在專案根目錄中建立名為 .ebextensions的目錄
2. 使用下列內容cloudwatch.config在 .ebextensions目錄中建立名為 的檔案：

```
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. 部署時，請在應用程式原始碼套件中包含 .ebextensions目錄

如需 Elastic Beanstalk 組態檔案的詳細資訊，請參閱使用組態檔案進行進階環境自訂。

遷移至 OpenTelemetry Java

本節提供從 X-Ray 開發套件遷移至適用於 Java 的 OpenTelemetry 開發套件應用程式的指引。

章節

- 零程式碼自動檢測解決方案
- 使用 SDK 的手動檢測解決方案
- 追蹤傳入請求（彈簧架構檢測）

- AWS SDK v2 檢測
- 檢測傳出的 HTTP 呼叫
- 其他程式庫的檢測支援
- 手動建立追蹤資料
- Lambda 檢測

零程式碼自動檢測解決方案

With X-Ray Java agent

若要啟用 X-Ray Java 代理程式，需要修改應用程式的 JVM 引數。

```
-javaagent:/path-to-disco/disco-java-agent.jar=pluginPath=/path-to-disco/disco-plugins
```

With OpenTelemetry-based Java agent

使用 OpenTelemetry 型 Java 代理程式。

- 使用 AWS Distro for OpenTelemetry (ADOT) Auto-Instrumentation Java 代理程式搭配 ADOT Java 代理程式進行自動檢測。如需詳細資訊，請參閱使用 Java 代理程式自動檢測追蹤和指標。如果您只想要追蹤，請停用 `OTEL_METRICS_EXPORTER=none` 環境變數。 從 Java 代理程式匯出指標。

  （選用）您也可以 AWS 在使用 ADOT Java 自動檢測功能自動檢測應用程式時啟用 CloudWatch Application Signals，以監控目前的應用程式運作狀態並追蹤長期應用程式效能。Application Signals 為您的應用程式、服務和相依性提供統一、以應用程式為中心的檢視，並協助監控和分類應用程式運作狀態。如需詳細資訊，請參閱 Application Signals。
- 使用 OpenTelemetry Java 代理程式進行自動檢測。如需詳細資訊，請參閱使用 Java Agent 進行零碼檢測。

使用 SDK 的手動檢測解決方案

Tracing setup with X-Ray SDK

若要使用適用於 Java 的 X-Ray 開發套件來檢測程式碼，首先需要使用服務外掛程式和本機取樣規則來設定 `AWSXRay`類別，然後使用提供的記錄器。

```
static { AWS XRayRecorderBuilder builder = AWS
 XRayRecorderBuilder.standard().withPlugin(new EC2Plugin()).withPlugin(new
 ECSPlugin()); AWS XRay.setGlobalRecorder(builder.build());
}
```

Tracing setup with OpenTelemetry SDK

需要下列相依性。

```
<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>
```

透過執行個體化 `TracerProvider`和全域註冊`OpenTelemetrySdk`物件來設定 OpenTelemetry SDK。設定這些元件：

- OTLP Span Exporter （例如 OtlpGrpcSpanExporter) - 將追蹤匯出至 CloudWatch 代理程式或 OpenTelemetry Collector 時需要
- AWS X-Ray 傳播器 – 將追蹤內容傳播至與 X-Ray 整合 AWS 的服務時需要
- AWS X-Ray 遠端取樣器 – 如果您需要使用 X-Ray 取樣規則來取樣請求，則為必要項目
- 資源偵測器 （例如 EcsResource 或 Ec2Resource) – 偵測執行您應用程式的主機中繼資料

```
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();
```

追蹤傳入請求 （彈簧架構檢測）

With X-Ray SDK

如需有關如何使用 X-Ray 開發套件搭配彈簧架構來檢測應用程式的資訊，請參閱搭配 Spring 的 AOP 和適用於 Java 的 X-Ray 開發套件。若要在 Spring 中啟用 AOP，請完成以下步驟。

1. 設定 Spring
2. 將追蹤篩選條件新增至您的應用程式
3. 註釋您的程式碼或實作 界面
4. 啟用應用程式中的 X-Ray

With OpenTelemetry SDK

OpenTelemetry 提供檢測程式庫，可收集 Spring Boot 應用程式傳入請求的追蹤。若要以最少的組態啟用 Spring Boot 檢測，請包含下列相依性。

```
<dependency>
           <groupId>io.opentelemetry.instrumentation</groupId>
            <artifactId>opentelemetry-spring-boot-starter</artifactId>
        </dependency>
```

如需如何為 OpenTelemetry 設定啟用和設定 Spring Boot 檢測的詳細資訊，請參閱 OpenTelemetry 入門。

Using OpenTelemetry-based Java agents

檢測 Spring Boot 應用程式的預設建議方法是使用 OpenTelemetry Java 代理程式搭配位元組碼檢測，與直接使用 SDK 相比，它還提供更多out-of-the-box檢測和組態。如需入門，請參閱 零程式碼自動檢測解決方案。

AWS SDK v2 檢測

With X-Ray SDK

當您在建置中新增`aws-xray-recorder-sdk-aws-sdk-v2-instrumentor`子模組時，適用於 Java 的 X-Ray 開發套件可以自動檢測所有 AWS SDK v2 用戶端。

若要使用適用於 Java 的 AWS SDK 2.2 和更新版本來檢測個別用戶端對 AWS 服務的下游用戶端呼叫，已排除建置組態中的`aws-xray-recorder-sdk-aws-sdk-v2-instrumentor` 模組，並包含`aws-xray-recorder-sdk-aws-sdk-v2`模組。透過使用 設定個別用戶端來進行檢測`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

若要自動檢測所有 AWS SDK 用戶端，請新增 `opentelemetry-aws-sdk-2.2-autoconfigure`子模組。

```
<dependency>
            <groupId>io.opentelemetry.instrumentation</groupId>
            <artifactId>opentelemetry-aws-sdk-2.2-autoconfigure</artifactId>
            <version>2.15.0-alpha</version>
            <scope>runtime</scope>
        </dependency>
```

若要檢測個別 AWS SDK 用戶端，請新增 `opentelemetry-aws-sdk-2.2`子模組。

```
<dependency>
            <groupId>io.opentelemetry.instrumentation</groupId>
            <artifactId>opentelemetry-aws-sdk-2.2</artifactId>
            <version>2.15.0-alpha</version>
            <scope>compile</scope>
        </dependency>
```

然後，在建立 AWS SDK 用戶端時註冊攔截器。

```
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();
```

檢測傳出的 HTTP 呼叫

With X-Ray SDK

若要使用 X-Ray 檢測傳出的 HTTP 請求，需要適用於 Java 的 X-Ray 開發套件的 Apache HttpClient 版本。

```
import com.amazonaws.xray.proxies.apache.http.HttpClientBuilder;
...
  public String randomName() throws IOException {
    CloseableHttpClient httpclient = HttpClientBuilder.create().build();
```

With OpenTelemetry SDK

與 X-Ray Java 開發套件類似，OpenTelemetry 提供的 `ApacheHttpClientTelemetry`類別具有建置器方法，允許建立 的執行個體`HttpClientBuilder`，為 Apache HttpClient 提供 OpenTelemetry 型跨度和內容傳播。

```
<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 opentelemetry-java-instrumentation 的程式碼範例。newHttpClient() 提供的 HTTP 用戶端將為執行的請求產生追蹤。

```
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();
  }
}
```

其他程式庫的檢測支援

在支援的程式庫、架構、應用程式伺服器和 JVM 下，在其個別的檢測 GitHub 儲存庫 中尋找 OpenTelemetry Java 支援的程式庫檢測的完整清單。 JVMs

或者，您可以搜尋 OpenTelemetry 登錄檔，了解 OpenTelemetry 是否支援檢測。若要開始搜尋，請參閱登錄檔。

手動建立追蹤資料

With X-Ray SDK

使用 X-Ray SDK，需要 `beginSegment`和 `beginSubsegment`方法來手動建立 X-Ray 區段和子區段。

```
 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

您可以使用自訂範圍來監控檢測程式庫未擷取的內部活動效能。請注意，只有跨度類型伺服器會轉換為 X-Ray 區段，所有其他跨度則會轉換為 X-Ray 子區段。

首先，您需要建立追蹤器來產生跨度，您可以透過 openTelemetry.getTracer方法取得。這將提供來自 的 Tracer 執行個體TracerProvider，該執行個體已在 使用 SDK 的手動檢測解決方案範例中全域註冊。您可以視需要建立任意數量的 Tracer 執行個體，但整個應用程式通常會有一個 Tracer。

```
Tracer tracer = openTelemetry.getTracer("my-app");
```

您可以使用 Tracer 來建立跨度。

```
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();
```

跨度的預設類型為 INTERNAL。

```
// 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 將註釋和中繼資料新增至追蹤

在上述範例中， `setAttribute`方法用於將屬性新增至每個跨度。根據預設，所有跨度屬性都會轉換為 X-Ray 原始資料中的中繼資料。為了確保屬性轉換為註釋而非中繼資料，上述範例會將該屬性的索引鍵新增至`aws.xray.annotations`屬性清單。如需詳細資訊，請參閱啟用自訂 X-Ray 註釋和註釋和中繼資料。

使用 OpenTelemetry 型 Java 代理程式

如果您使用 Java 代理程式自動檢測應用程式，則需要在應用程式中執行手動檢測。例如，針對任何自動檢測程式庫未涵蓋的區段，在應用程式中檢測程式碼。

若要使用代理程式執行手動檢測，您需要使用`opentelemetry-api` 成品。成品版本不能比代理程式版本更新。

```
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 檢測

With X-Ray SDK

使用 X-Ray 開發套件，在 Lambda 啟用主動追蹤之後，使用 X-Ray 開發套件不需要額外的組態。Lambda 將建立代表 Lambda 處理常式調用的區段，而且您可以使用 X-Ray SDK 建立子區段或儀器程式庫，而不需要任何額外的組態。

With OpenTelemetry-based solutions

自動檢測 Lambda 圖層 – 您可以使用下列解決方案，自動檢測 Lambda 與已 AWS 佈建的 Lambda 圖層：

- CloudWatch Application Signals Lambda layer（建議）

> **Note**
>
> 此 Lambda 層預設啟用 CloudWatch Application Signals，可透過收集指標和追蹤來啟用 Lambda 應用程式的效能和運作狀態監控。若只要追蹤，請設定 Lambda 環境變數 `OTEL_AWS_APPLICATION_SIGNALS_ENABLED=false`。

  - 啟用 Lambda 應用程式的效能和運作狀態監控
  - 根據預設，同時收集指標和追蹤
- AWS ADOT Java 的 受管 Lambda 層。如需詳細資訊，請參閱 AWS Distro for OpenTelemetry Lambda Support for Java。

若要搭配自動檢測層使用手動檢測，請參閱 使用 SDK 的手動檢測解決方案。為了減少冷啟動，請考慮使用 OpenTelemetry 手動檢測為您的 Lambda 函數產生 OpenTelemetry 追蹤。

適用於 AWS Lambda 的 OpenTelemetry 手動檢測

請考慮下列進行 Amazon S3 ListBuckets 呼叫的 Lambda 函數程式碼。

```
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);
        }
    }
}
```

以下是相依性。

```
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')
}
```

若要手動檢測 Lambda 處理常式和 Amazon S3 用戶端，請執行下列動作。

1. 將實作 RequestHandler（或 RequestStreamHandler) 的函數類別取代為延伸 TracingRequestHandler（或 TracingRequestStreamHandler) 的函數類別。
2. 執行個體化 TracerProvider 並全域註冊 OpenTelemetrySdk 物件。建議將 TracerProvider 設定為：
   a. 具有 X-Ray UDP 跨度匯出程式的簡易跨度處理器，可將追蹤傳送至 Lambda 的 UDP X-Ray 端點
   b. ParentBased always on sampler （若未設定，則預設為預設值）
   c. 將 service.name 設為 Lambda 函數名稱的資源
   d. X-Ray Lambda 傳播器
3. 將 handleRequest 方法變更為 ，doHandleRequest並將OpenTelemetrySdk物件傳遞至 基礎類別。
4. 透過在建置用戶端時註冊攔截器，使用 OpenTemetry AWS SDK 檢測來檢測 Amazon S3 用戶端。

您需要下列 OpenTelemetry 相關相依性。

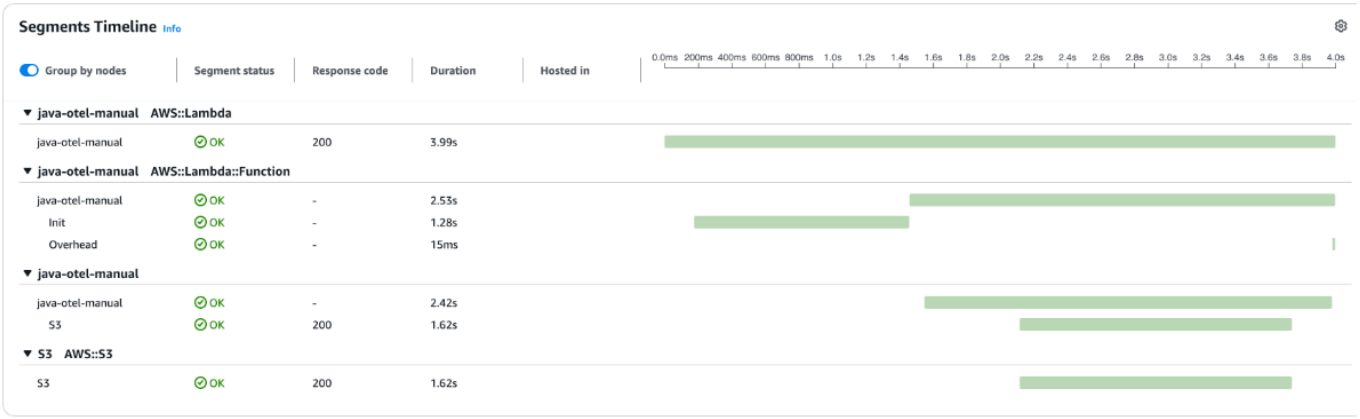
```
dependencies {
    ...

    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')
}
```

下列程式碼示範必要的變更之後的 Lambda 函數。您可以建立額外的自訂範圍，以補充自動提供的範圍。

```
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);
        }
    }
}
```

叫用 Lambda 函數時，您會在 CloudWatch 主控台的追蹤地圖下看到下列追蹤。



遷移至 OpenTelemetry Go

從 X-Ray 遷移時，請使用下列程式碼範例，使用 OpenTelemetry SDK 手動檢測 Go 應用程式。

使用 SDK 手動檢測

Tracing setup with X-Ray SDK

使用適用於 Go 的 X-Ray 開發套件時，在檢測程式碼之前，需要設定服務外掛程式或本機取樣規則。

```
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

透過執行個體化 TracerProvider 並將其註冊為全域追蹤器供應商，來設定 OpenTelemetry SDK。我們建議您設定下列元件：

- OTLP 追蹤匯出工具 – 將追蹤匯出至 CloudWatch Agent 或 OpenTelemetry Collector 時需要
- X-Ray 傳播器 – 將追蹤內容傳播至與 X-Ray 整合 AWS 的服務時需要
- X-Ray 遠端取樣器 – 使用 X-Ray 取樣規則取樣請求時需要
- 資源偵測器 – 偵測執行您應用程式的主機中繼資料

```
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
}
```

追蹤傳入請求 (HTTP 處理常式檢測)

With X-Ray SDK

為了使用 X-Ray 檢測 HTTP 處理常式，X-Ray 處理常式方法用於使用 NewFixedSegmentNamer 產生區段。

```
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

若要使用 OpenTelemetry 檢測 HTTP 處理常式，請使用 OpenTelemetry 的 newHandler 方法來包裝原始處理常式程式碼。

```
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 適用於 Go v2 的 SDK 檢測

With X-Ray SDK

若要檢測來自 AWS SDK 的傳出 AWS 請求，您的用戶端會依下列方式進行檢測：

```
// 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

OpenTelemetry 的 AWS SDK for Go v2 Instrumentation 提供下游 AWS SDK 呼叫的追蹤支援。以下是追蹤 S3 用戶端呼叫的範例：

```
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
    }
```

檢測傳出的 HTTP 呼叫

With X-Ray SDK

為了使用 X-Ray 檢測傳出的 HTTP 呼叫，xray.Client 用於建立所提供 HTTP 用戶端的副本。

```
myClient := xray.Client(http-client)

resp, err := ctxhttp.Get(ctx, xray.Client(nil), url)
```

With OpenTelemetry SDK

若要使用 OpenTelemetry 檢測 HTTP 用戶端，請使用 OpenTelemetry 的 otelhttp.NewTransport 方法來包裝 http.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)
```

其他程式庫的檢測支援

您可以在檢測套件 下找到 OpenTelemetry Go 支援程式庫檢測的完整清單。 https://github.com/open-telemetry/opentelemetry-go-contrib/tree/main/instrumentation#instrumentation-packages

或者，您可以搜尋 OpenTelemetry 登錄檔，了解 OpenTelemetry 是否支援對登錄檔下的程式庫進行檢測。

手動建立追蹤資料

With X-Ray SDK

使用 X-Ray 開發套件時，需要 BeginSegment 和 BeginSubsegment 方法，才能手動建立 X-Ray 區段和子區段。

```
// 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

使用自訂範圍來監控檢測程式庫未擷取的內部活動效能。請注意，只有種類的伺服器會轉換為 X-Ray 區段，所有其他範圍則會轉換為 X-Ray 子區段。

首先，您需要建立追蹤器來產生跨度，您可以透過 `otel.Tracer`方法取得。這將提供來自 TracerProvider 的 Tracer 執行個體，該執行個體已在追蹤設定範例中全域註冊。您可以視需要建立任意數量的 Tracer 執行個體，但整個應用程式通常會有一個 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 將註釋和中繼資料新增至追蹤

在上述範例中，`WithAttributes`方法用於將屬性新增至每個跨度。請注意，根據預設，所有跨度屬性都會轉換為 X-Ray 原始資料中的中繼資料。若要確保屬性轉換為註釋而非中繼資料，請將屬性的索引鍵新增至`aws.xray.annotations`屬性清單。如需詳細資訊，請參閱啟用自訂 X-Ray 註釋。

Lambda 手動檢測

With X-Ray SDK

使用 X-Ray 開發套件時，在 Lambda 啟用主動追蹤之後，不需要其他組態即可使用 X-Ray 開發套件。Lambda 建立了代表 Lambda 處理常式調用的區段，而且您使用 X-Ray 開發套件建立子區段，而不需要任何其他組態。

With OpenTelemetry SDK

下列 Lambda 函數程式碼（不含檢測）會發出 Amazon S3 ListBuckets 呼叫和傳出 HTTP 請求。

```
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)
}
```

若要手動檢測 Lambda 處理常式和 Amazon S3 用戶端，請執行下列動作：

1. 在 main() 中，執行個體化 TracerProvider (tp)，並將其註冊為全域追蹤器提供者。建議將 TracerProvider 設定為：
   a. 具有 X-Ray UDP 跨度匯出程式的簡易跨度處理器，可將追蹤傳送至 Lambda 的 UDP X-Ray 端點
   b. 將 service.name 設定為 Lambda 函數名稱的資源
2. 將 的用量`lambda.Start(lambdaHandler)`變更為 `lambda.Start(otellambda.InstrumentHandler(lambdaHandler, xrayconfig.WithRecommendedOptions(tp)...))`。
3. 透過將 的 OpenTelemetry 中介軟體附加`aws-sdk-go-v2`至 Amazon S3 用戶端組態，使用 OpenTemetry SDK 檢測來檢測 Amazon S3 用戶端。 OpenTemetry AWS OpenTelemetry
4. 使用 OpenTelemetry 的 `otelhttp.NewTransport`方法來包裝 ，以檢測 http 用戶端`http.DefaultTransport`。

下列程式碼是 Lambda 函數在變更後的外觀範例。除了自動提供的範圍之外，您還可以手動建立其他自訂範圍。

```
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)...))
}
```

叫用 Lambda 時，您會在 CloudWatch 主控台的 Trace Map中看到下列追蹤：

遷移至 OpenTelemetry Node.js

本節說明如何將 Node.js 應用程式從 X-Ray SDK 遷移至 OpenTelemetry。它涵蓋自動和手動檢測方法，並提供常見使用案例的特定範例。

X-Ray Node.js SDK 可協助您手動檢測 Node.js 應用程式以進行追蹤。本節提供從 X-Ray 遷移到 OpenTelemetry 檢測的程式碼範例。

章節

零程式碼自動檢測解決方案

若要使用適用於 Node.js 的 X-Ray 開發套件追蹤請求，您必須修改應用程式碼。使用 OpenTelemetry，您可以使用零碼自動檢測解決方案來追蹤請求。

使用 OpenTelemetry 型自動檢測的零程式碼自動檢測。

1. 使用適用於 Node.js 的 AWS Distro for OpenTelemetry (ADOT) 自動檢測 – 如需 Node.js 應用程式的自動檢測，請參閱使用適用於 OpenTelemetry JavaScript 自動檢測的 AWS Distro 追蹤和指標。

   （選用）您也可以 AWS 在使用 ADOT JavaScript 自動檢測功能自動檢測應用程式時啟用 CloudWatch Application Signals，以監控目前的應用程式運作狀態，並根據業務目標追蹤長期應用程式效能。Application Signals 為您提供應用程式、服務和相依性的統一、以應用程式為中心的檢視，並協助您監控和分類應用程式運作狀態。如需詳細資訊，請參閱 Application Signals。
2. 使用 OpenTelemetry JavaScript 零碼自動檢測 – 如需使用 OpenTelemetry JavaScript 自動檢測，請參閱 JavaScript 零碼檢測。

手動檢測解決方案

Tracing setup with X-Ray SDK

使用適用於 Node.js 的 X-Ray 開發套件時，需要使用 `aws-xray-sdk`套件來設定 X-Ray 開發套件搭配服務外掛程式或本機取樣規則，然後再使用 開發套件來檢測您的程式碼。

```
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 X-Ray 遠端取樣目前無法針對 OpenTelemetry JS 進行設定。不過，目前可透過適用於 Node.js 的 ADOT Auto-Instrumentation 支援 X-Ray 遠端取樣。

針對下列程式碼範例，您將需要下列相依性：

```
npm install --save \
  @opentelemetry/api \
```

```
  @opentelemetry/sdk-node \
  @opentelemetry/exporter-trace-otlp-proto \
  @opentelemetry/propagator-aws-xray \
  @opentelemetry/resource-detector-aws
```

您必須先設定 OpenTelemetry SDK，才能執行應用程式程式碼。這可以透過使用 --require 旗標來完成。建立名為 instrumentation.js 的檔案，其中包含您的 OpenTelemetry 檢測組態和設定。

建議您設定下列元件：

- OTLPTraceExporter – 將追蹤匯出至 CloudWatch Agent/OpenTelemetry Collector 時需要
- AWSXRayPropagator – 將追蹤內容傳播至與 X-Ray 整合 AWS 的服務時需要
- Resource Detectors （例如 Amazon EC2 Resource Detector) - 偵測執行您應用程式的主機中繼資料

```
/*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();
```

然後，您可以使用 OpenTelemetry 設定來執行應用程式，例如：

```
node --require ./instrumentation.js app.js
```

您可以使用 OpenTelemetry SDK 程式庫檢測，自動建立 AWS SDK 等程式庫的範圍。啟用這些項目會自動建立模組的範圍，例如適用於 JavaScript 的 AWS SDK v3。OpenTelemetry 提供啟用所有程式庫檢測的選項，或指定要啟用哪些程式庫檢測。

若要啟用所有檢測，請安裝 `@opentelemetry/auto-instrumentations-node`套件：

```
npm install @opentelemetry/auto-instrumentations-node
```

接著，更新組態以啟用所有程式庫檢測，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

您可以使用適用於 Node.js 的 ADOT 自動檢測，為您的 Node.js 應用程式自動設定 OpenTelemetry。透過使用 ADOT Auto-Instrumentation，您不需要手動變更程式碼來追蹤傳入的請求，或追蹤 AWS SDK 或 HTTP 用戶端等程式庫。如需詳細資訊，請參閱使用 AWS Distro for OpenTelemetry JavaScript Auto-Instrumentation 追蹤 和 指標。

Node.js 的 ADOT 自動檢測支援：

- 透過環境變數的 X-Ray 遠端取樣 – `export OTEL_TRACES_SAMPLER=xray`

- X-Ray 追蹤內容傳播 （預設為啟用）
- 資源偵測 (Amazon EC2、Amazon ECS 和 Amazon EKS 環境的資源偵測預設為啟用）
- 所有支援的 OpenTelemetry 檢測的自動程式庫檢測，可透過 `OTEL_NODE_ENABLED_INSTRUMENTATIONS`和 `OTEL_NODE_DISABLED_INSTRUMENTATIONS` 環境變數選擇性地停用/啟用
- 手動建立跨度

追蹤傳入的請求

With X-Ray SDK

Express.js

使用 X-Ray 開發套件追蹤 Express.js 應用程式收到的傳入 HTTP 請求，`AWSXRay.express.closeSegment()`需要兩個中介軟體`AWSXRay.express.openSegment(<name>)`和 來包裝所有定義的路由，以便追蹤它們。

```
app.use(xrayExpress.openSegment('defaultName'));

...

app.use(xrayExpress.closeSegment());
```

重新設定

若要追蹤`Restify`應用程式收到的傳入 HTTP 請求，透過從 Restify Server 上的 `aws-xray-sdk-restify` 模組執行啟用，使用 X-Ray 開發套件的中介軟體：

```
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

追蹤 的傳入請求支援Express.js是由 OpenTelemetry HTTP 檢測和 OpenTelemetry Express 檢測提供。使用 安裝下列相依性npm：

```
npm install --save @opentelemetry/instrumentation-http @opentelemetry/
instrumentation-express
```

更新 OpenTelemetry SDK 組態以啟用快速模組的檢測：

```
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(),
  ],
});
```

重新設定

對於 Restify 應用程式，您將需要 OpenTelemetry Restify 檢測。安裝下列相依性：

```
npm install --save @opentelemetry/instrumentation-restify
```

更新 OpenTelemetry SDK 組態以啟用 Restify 模組的檢測：

```
const { RestifyInstrumentation } = require('@opentelemetry/instrumentation-
restify');
...

const sdk = new NodeSDK({
  ...

  instrumentations: [
    ...
    new RestifyInstrumentation(),
  ],
});
```

AWS SDK JavaScript V3 檢測

With X-Ray SDK

若要檢測來自 AWS SDK 的傳出 AWS 請求，您檢測了用戶端，如下列範例所示：

```
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

追蹤對 DynamoDB、Amazon S3 和其他 的下游 AWS SDK 呼叫支援，是由 OpenTelemetry AWS SDK 檢測提供。使用 安裝下列相依性npm：

```
npm install --save @opentelemetry/instrumentation-aws-sdk
```

使用 SDK 檢測更新 OpenTelemetry AWS SDK 組態。

```
import { AwsInstrumentation } from '@opentelemetry/instrumentation-aws-sdk';
...

const sdk = new NodeSDK({
  ...

  instrumentations: [
    ...
    new AwsInstrumentation()
  ],
});
```

檢測傳出的 HTTP 呼叫

With X-Ray SDK

若要使用 X-Ray 檢測傳出的 HTTP 請求，需要檢測用戶端。例如，請參閱以下。

個別 HTTP 用戶端

```
var AWSXRay = require('aws-xray-sdk');
var http = AWSXRay.captureHTTPs(require('http'));
```

所有 HTTP 用戶端（全域）

```
var AWSXRay = require('aws-xray-sdk');
AWSXRay.captureHTTPsGlobal(require('http'));
var http = require('http');
```

With OpenTelemetry SDK

追蹤 Node.js HTTP 用戶端的支援是由 OpenTelemetry HTTP 檢測提供。使用 安裝下列相依性npm：

```
npm install --save @opentelemetry/instrumentation-http
```

更新 OpenTelemetry SDK 組態，如下所示：

```
const { HttpInstrumentation } = require('@opentelemetry/instrumentation-http');
...

const sdk = new NodeSDK({
  ...

  instrumentations: [
    ...
    new HttpInstrumentation(),
  ],
});
```

其他程式庫的檢測支援

您可以在支援的檢測 下找到 OpenTelemetry JavaScript 支援的程式庫檢測的完整清單。

或者，您可以搜尋 OpenTelemetry 登錄檔，了解 OpenTelemetry 是否支援在登錄檔下檢測您的程式庫。

手動建立追蹤資料

With X-Ray SDK

使用 X-Ray 時，需要`aws-xray-sdk`套件程式碼來手動建立區段及其子區段，以追蹤您的應用程式。

```
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

您可以建立並使用自訂範圍來監控檢測程式庫未擷取的內部活動效能。請注意，只有種類的伺服器會轉換為 X-Ray 區段，所有其他範圍則會轉換為 X-Ray 子區段。如需詳細資訊，請參閱客群。

在追蹤設定中設定 OpenTelemetry SDK 以建立跨度之後，您將需要一個 Tracer 執行個體。您可以視需要建立任意數量的 Tracer 執行個體，但整個應用程式通常會有一個 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 將註釋和中繼資料新增至追蹤

您也可以將自訂鍵/值對新增為跨度中的屬性。請注意，根據預設，所有這些跨度屬性都會轉換為 X-Ray 原始資料中的中繼資料。若要確保屬性轉換為註釋而非中繼資料，請將屬性的索引鍵新增至`aws.xray.annotations`屬性清單。如需詳細資訊，請參閱[啟用自訂 X-Ray 註釋](#)。

```
 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 檢測

With X-Ray SDK

為 Lambda 函數啟用 Active Tracing 後，需要 X-Ray 開發套件，無需額外的組態。Lambda 會建立代表 Lambda 處理常式調用的區段，而且您使用 X-Ray SDK 建立子區段或檢測程式庫，而不需要任何其他組態。

With OpenTelemetry SDK

您可以使用 AWS 付費 Lambda 層自動檢測 Lambda。有兩種解決方案：

- （建議）CloudWatch Application Signals lambda layer

> **Note**
>
> 此 Lambda 層預設啟用 CloudWatch Application Signals，可透過收集指標和追蹤來啟用 Lambda 應用程式的效能和運作狀態監控。如果您只想要追蹤，您應該設定 Lambda 環境變數 `OTEL_AWS_APPLICATION_SIGNALS_ENABLED=false`。如需詳細資訊，請參閱在 [Lambda 上啟用您的應用程式](#)。

- AWS ADOT JS 的 受管 Lambda 層。如需詳細資訊，請參閱 AWS Distro for OpenTelemetry Lambda Support for JavaScript。

使用 Lambda 檢測手動建立 Spans

雖然 ADOT JavaScript Lambda Layer 為您的 Lambda 函數提供自動檢測，但您可能會發現需要在 Lambda 中執行手動檢測，例如，提供自訂資料或在程式庫檢測未涵蓋的 Lambda 函數本身內檢測程式碼。

若要搭配自動檢測執行手動檢測，您需要新增 `@opentelemetry/api`做為相依性。此相依性的版本建議與 ADOT JavaScript SDK 使用的相同相依性版本相同。您可以使用 OpenTelemetry API 在 Lambda 函數中手動建立跨度。

若要使用 NPM 新增`@opentelemetry/api`相依性：

```
npm install @opentelemetry/api
```

遷移至 OpenTelemetry .NET

在 .NET 應用程式中使用 X-Ray 追蹤時，手動檢測會使用 X-Ray .NET 開發套件。

本節提供 使用 SDK 的手動檢測解決方案區段中的程式碼範例，用於從 X-Ray 手動檢測解決方案遷移至適用於 .NET 的 OpenTelemetry 手動檢測解決方案。或者，您可以從 X-Ray 手動檢測遷移到 OpenTelemetry 自動檢測解決方案到檢測 .NET 應用程式，而無需修改 零程式碼自動檢測解決方案區段中的應用程式原始碼。

章節

- 零程式碼自動檢測解決方案
- 使用 SDK 的手動檢測解決方案
- 手動建立追蹤資料
- 追蹤傳入的請求 (ASP.NET 和 ASP.NET 核心檢測)
- AWS SDK 檢測
- 檢測傳出的 HTTP 呼叫
- 其他程式庫的檢測支援

- Lambda 檢測

零程式碼自動檢測解決方案

OpenTelemetry 提供零程式碼自動檢測解決方案。這些解決方案會追蹤請求，而不需要變更您的應用程式程式碼。

OpenTelemetry 型自動檢測選項

1. 使用適用於 .NET 的 AWS Distro for OpenTelemetry (ADOT) 自動檢測 – 若要自動檢測 .NET 應用程式，請參閱使用適用於 OpenTelemetry 的 AWS Distro .NET Auto-Instrumentation 追蹤和指標。

   （選用） AWS 使用 ADOT .NET 自動檢測在 上自動檢測應用程式時啟用 CloudWatch Application Signals，以：

   - 監控目前的應用程式運作狀態
   - 根據業務目標追蹤長期應用程式效能
   - 取得應用程式、服務和相依性的統一、以應用程式為中心的檢視
   - 監控和分類應用程式運作狀態

   如需詳細資訊，請參閱 Application Signals。
2. 使用 OpenTelemetry .Net 零碼自動檢測 – 若要使用 OpenTelemetry .Net 自動檢測，請參閱使用 AWS Distro for OpenTelemetry .NET Auto-Instrumentation 追蹤和指標。

使用 SDK 的手動檢測解決方案

Tracing configuration with X-Ray SDK

對於 .NET Web 應用程式，X-Ray SDK 是在 `Web.config` 檔案的 appSettings 區段中設定。

範例 Web.config

```
<configuration>
  <appSettings>
    <add key="AWSXRayPlugins" value="EC2Plugin"/>
  </appSettings>
</configuration>
```

對於 .NET Core，XRay會使用名為 appsettings.json且具有最上層金鑰的檔案，然後建置組態物件來初始化 X-Ray 記錄器。

.NET 的範例 appsettings.json

```
{
  "XRay": {
    "AWSXRayPlugins": "EC2Plugin"
  }
}
```

.NET Core Program.cs – 記錄器組態的範例

```
using Amazon.XRay.Recorder.Core;
...
AWSXRayRecorder.InitializeInstance(configuration);
```

Tracing configuration with OpenTelemetry SDK

新增這些相依性：

```
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
```

針對您的 .NET 應用程式，請設定 Global TracerProvider 來設定 OpenTelemetry SDK。下列範例組態也會啟用 的檢測ASP.NET Core。若要檢測 ASP.NET，請參閱 追蹤傳入的請求 (ASP.NET 和 ASP.NET 核心檢測) 。若要搭配其他架構使用 OpenTelemetry，請參閱 登錄檔以取得更多支援架構的程式庫。

建議您設定下列元件：

- An `OTLP Exporter` – 將追蹤匯出至 CloudWatch Agent/OpenTelemetry Collector 時需要
- AWS X-Ray 傳播器 – 將追蹤內容傳播至AWS 與 X-Ray 整合的服務時需要
- AWS X-Ray 遠端取樣器 – 如果您需要使用 X-Ray 取樣規則來取樣請求，則為必要項目
- `Resource Detectors` （例如，Amazon EC2 Resource Detector) - 偵測執行您應用程式的主機中繼資料

```
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
```

若要將 OpenTelemetry 用於主控台應用程式，請在程式啟動時新增下列 OpenTelemetry 組態。

```
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
```

手動建立追蹤資料

With X-Ray SDK

使用 X-Ray 開發套件，需要 `BeginSegment`和 `BeginSubsegment`方法來手動建立 X-Ray 區段和子區段。

```
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，您可以使用活動 API 來建立自訂範圍，以監控檢測程式庫未擷取的內部活動效能。請注意，只有種類的伺服器會轉換為 X-Ray 區段，所有其他範圍則會轉換為 X-Ray 子區段。

您可以視需要建立任意數量的ActivitySource執行個體，但建議在整個應用程式/服務中只有一個執行個體。

```
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 將註釋和中繼資料新增至追蹤

您也可以在 活動上使用 SetTag方法，將自訂金鑰/值對做為屬性新增至您的範圍。請注意，根據預設，所有跨度屬性都會轉換為 X-Ray 原始資料中的中繼資料。若要確保屬性轉換為註釋而非中繼資料，您可以將該屬性的索引鍵新增至aws.xray.annotations屬性清單。

```
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 自動檢測

如果您使用適用於 .NET 的 OpenTelemetry 自動檢測解決方案，而且如果您需要在應用程式中執行手動檢測，例如，針對任何自動檢測程式庫未涵蓋的區段，在應用程式本身內檢測程式碼。

由於只能有一個全域 `TracerProvider`，`TracerProvider`如果與自動檢測一起使用，則手動檢測不應執行個體化自己的 。`TracerProvider` 使用 時，透過 OpenTelemetry SDK 使用自動檢測或手動檢測時，自訂手動追蹤的運作方式相同。

追蹤傳入的請求 (ASP.NET 和 ASP.NET 核心檢測)

With X-Ray SDK

若要檢測 ASP.NET 應用程式提供的請求，請參閱 以取得如何在 `global.asax` 檔案的 `Init`方法`RegisterXRay`中呼叫 https://docs.aws.amazon.com/xray/latest/devguide/xray-sdk-dotnet-messagehandler.html 的資訊。

```
AWSXRayASPNET.RegisterXRay(this, "MyApp");
```

若要檢測 ASP.NET 核心應用程式提供的請求，會在啟動類別`UseXRay`方法中的任何其他中介軟體之前呼叫 `Configure`方法。

```
app.UseXRay("MyApp");
```

With OpenTelemetry SDK

OpenTelemetry 也提供檢測程式庫，以收集 ASP.NET 和 ASP.NET 核心傳入 Web 請求的追蹤。下一節列出為您的 OpenTelemetry 組態新增和啟用這些程式庫檢測所需的步驟，包括如何在建立 Tracer Provider 時新增 ASP.NET 或 ASP.NET 核心檢測。

如需如何啟用 OpenTelemetry.Instrumentation.AspNet,請參閱啟用 OpenTelemetry.Instrumentation.AspNet 的步驟，以及有關如何啟用 OpenTelemetry.Instrumentation.AspNetCore,請參閱啟用 OpenTelemetry.Instrumentation.AspNetCore 的步驟。

AWS SDK 檢測

With X-Ray SDK

呼叫 安裝所有 AWS SDK 用戶端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
```

使用下列其中一種方法來進行特定 AWS 服務用戶端檢測。

```
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

針對下列程式碼範例，您將需要下列相依性：

```
dotnet add package OpenTelemetry.Instrumentation.AWS
```

若要檢測 AWS SDK，請更新設定 Global TracerProvider 的 OpenTelemetry SDK 組態。

```
builder.Services.AddOpenTelemetry()
    ...
    .WithTracing(tracing => tracing
        .AddAWSInstrumentation()
        ...
```

檢測傳出的 HTTP 呼叫

With X-Ray SDK

X-Ray .NET SDK 透過擴充功能方法`GetResponseTraced()`或使用`GetAsyncResponseTraced()`時`System.Net.HttpWebRequest`，或使用 `HttpClientXRayTracingHandler` 處理常式來追蹤傳出的 HTTP 呼叫`System.Net.Http.HttpClient`。

With OpenTelemetry SDK

針對下列程式碼範例，您將需要下列相依性：

```
dotnet add package OpenTelemetry.Instrumentation.Http
```

若要檢測 `System.Net.Http.HttpClient`和 `System.Net.HttpWebRequest`，請更新設定 Global TracerProvider 的 OpenTelemetry SDK 組態。

```
builder.Services.AddOpenTelemetry()
    ...
    .WithTracing(tracing => tracing
        .AddHttpClientInstrumentation()
        ...
```

其他程式庫的檢測支援

您可以搜尋和篩選適用於 .NET 檢測程式庫的 OpenTelemetry 登錄檔，以了解 OpenTelemetry 是否支援您的程式庫檢測。請參閱 登錄檔以開始搜尋。

Lambda 檢測

With X-Ray SDK

使用 X-Ray 開發套件搭配 Lambda 需要下列程序：

1. 在 Lambda 函數上啟用主動追蹤

2. Lambda 服務會建立代表處理常式調用的客群
3. 使用 X-Ray SDK 建立子區段或檢測程式庫

With OpenTelemetry-based solutions

您可以使用 AWS 付費的 Lambda 層自動檢測 Lambda。有兩種解決方案：

- （建議）CloudWatch Application Signals lambda layer
- 為了獲得更好的效能，建議您考慮使用 OpenTelemetry Manual Instrumentation 為 Lambda 函數產生 OpenTelemetry 追蹤。

適用於 AWS Lambda 的 OpenTelemetry 手動檢測

以下是 Lambda 函數程式碼（不含檢測）範例。

```
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);
        }
    }
}
```

若要手動檢測 Lambda 處理常式和 Amazon S3 用戶端，請執行下列動作。

1. 執行個體化 TracerProvider – 建議將 TracerProvider 設定為 `XrayUdpSpanExporter`、ParentBased Always On 取樣器，以及將 `Resourceservice.name`設定為 Lambda 函數名稱的 。
2. 呼叫 將 SDK 用戶端檢測新增至 ，以使用 OpenTemetry AWS AWS SDK 檢測來檢測 `AddAWSInstrumentation()` Amazon S3 用戶端 `TracerProvider`
3. 建立與原始 Lambda 函數具有相同簽章的包裝函式。呼叫 `AWSLambdaWrapper.Trace()` API 並傳遞 `TracerProvider`、原始 Lambda 函數及其輸入做為參數。將包裝函式設定為 Lambda 處理常式輸入。

針對下列程式碼範例，您將需要下列相依性：

```
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
```

下列程式碼示範必要的變更之後的 Lambda 函數。您可以建立額外的自訂範圍，以補充自動提供的範圍。

```
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);
        }
    }
}
```

叫用此 Lambda 時，您會在 CloudWatch 主控台的追蹤地圖中看到下列追蹤：

遷移至 OpenTelemetry Python

本指南可協助您將 Python 應用程式從 X-Ray SDK 遷移至 OpenTelemetry 檢測。它涵蓋自動和手動檢測方法，並提供常見案例的程式碼範例。

章節

- 零程式碼自動檢測解決方案
- 手動檢測您的應用程式
- 追蹤設定初始化
- 追蹤傳入的請求
- AWS SDK 檢測
- 透過請求檢測傳出的 HTTP 呼叫
- 其他程式庫的檢測支援
- 手動建立追蹤資料
- Lambda 檢測

零程式碼自動檢測解決方案

使用 X-Ray SDK，您必須修改應用程式程式碼以追蹤請求。OpenTelemetry 提供零程式碼的自動檢測解決方案來追蹤請求。使用 OpenTelemetry，您可以選擇使用零碼自動檢測解決方案來追蹤請求。

使用 OpenTelemetry 型自動檢測的零代碼

1. 使用適用於 Python 的 AWS Distro for OpenTelemetry (ADOT) 自動檢測 – 如需適用於 Python 應用程式的自動檢測，請參閱使用適用於 OpenTelemetry Python 自動檢測的 AWS Distro 追蹤和指標。

（選用）您也可以 AWS 在使用 ADOT Python 自動檢測功能自動檢測應用程式時啟用 CloudWatch Application Signals，以監控目前的應用程式運作狀態，並根據業務目標追蹤長期應用程式效能。Application Signals 為您提供應用程式、服務和相依性的統一、以應用程式為中心的檢視，並協助您監控和分類應用程式運作狀態。

2. 使用 OpenTelemetry Python 零碼自動檢測 – 如需使用 OpenTelemetry Python 自動檢測，請參閱 Python 零碼檢測。

手動檢測您的應用程式

您可以使用 `pip`命令手動檢測您的應用程式。

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
```

追蹤設定初始化

With X-Ray SDK

在 X-Ray 中，全域`xray_recorder`會初始化並用來產生區段和子區段。

With OpenTelemetry SDK

> **Note**
>
> X-Ray 遠端取樣目前無法針對 OpenTelemetry Python 進行設定。不過，目前可透過 ADOT Auto-Instrumentation for Python 支援 X-Ray 遠端取樣。

在 OpenTelemetry 中，您需要初始化全域 `TracerProvider`。使用此 `TracerProvider`，您可以取得 Tracer，用於在應用程式中任何位置產生範圍。建議您設定下列元件：

- `OTLPSpanExporter` – 將追蹤匯出至 CloudWatch Agent/OpenTelemetry Collector 時需要
- AWS X-Ray 傳播器 – 將追蹤內容傳播至與 X-Ray 整合 AWS 的服務時需要

```
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
```

使用適用於 Python 的 ADOT 自動檢測

您可以使用適用於 Python 的 ADOT 自動檢測，為您的 Python 應用程式自動設定 OpenTelemetry。透過使用 ADOT 自動檢測，您不需要手動變更程式碼來追蹤傳入的請求，或追蹤 AWS SDK 或 HTTP 用戶端等程式庫。如需詳細資訊，請參閱使用 AWS Distro for OpenTelemetry Python Auto-Instrumentation 追蹤 和 指標。

適用於 Python 的 ADOT 自動檢測支援：

- 透過環境變數進行 X-Ray 遠端取樣 `export OTEL_TRACES_SAMPLER=xray`
- X-Ray 追蹤內容傳播 （ 預設為啟用 ）
- 資源偵測 (Amazon EC2、Amazon ECS 和 Amazon EKS 環境的資源偵測預設為啟用)
- 預設會啟用所有支援的 OpenTelemetry 檢測的自動程式庫檢測。您可以透過`OTEL_PYTHON_DISABLED_INSTRUMENTATIONS` 環境變數選擇性地停用 （ 預設為啟用 ）
- 手動建立跨度

從 X-Ray 服務外掛程式到 OpenTelemetry AWS 資源提供者

X-Ray 開發套件提供外掛程式，您可以新增至 `xray_recorder`，以從 Amazon EC2、Amazon ECS 和 Elastic Beanstalk 等託管服務擷取平台特定資訊。它類似於 OpenTelemetry 中將資訊擷取為資源屬性的資源提供者。有多個資源提供者可用於不同的 AWS 平台。

- 首先安裝 AWS 延伸套件， `pip install opentelemetry-sdk-extension-aws`
- 設定所需的資源偵測器。下列範例示範如何在 OpenTelemetry SDK 中設定 Amazon EC2 資源提供者

```
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)
```

追蹤傳入的請求

With X-Ray SDK

X-Ray Python SDK 支援應用程式架構，例如 Django、Flask 和 Bottle，以追蹤在其上執行的 Python 應用程式傳入請求。方法是將 `XRayMiddleware`新增至每個架構的應用程式。

With OpenTelemetry SDK

OpenTelemetry 透過特定的檢測程式庫提供 Django 和 Flask 的檢測。OpenTelemetry 中沒有可用的 Bottle 檢測，仍然可以使用 OpenTelemetry WSGI 檢測來追蹤應用程式。

針對下列程式碼範例，您需要下列相依性：

```
pip install opentelemetry-instrumentation-flask
```

您必須先初始化 OpenTelemetry SDK 並註冊全域 TracerProvider，才能為您的應用程式架構新增檢測。如果沒有它，追蹤操作將是 `no-ops`。設定全域 之後`TracerProvider`，您就可以針對應用程式架構使用 檢測器。下列範例示範 Flask 應用程式。

```
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 SDK 檢測

With X-Ray SDK

X-Ray Python SDK 透過修補程式`botocore`庫來追蹤 AWS SDK 用戶端請求。如需詳細資訊，請參閱使用適用於 Python 的 X-Ray AWS SDK 追蹤 SDK 呼叫。在您的應用程式中，`patch_all()`方法用於使用 或 程式庫，選擇性地檢測所有`boto3`程式庫`botocore`或修補程式`patch(['botocore'])`。任何選擇的方法都會檢測應用程式中的所有 Boto3 用戶端，並針對使用這些用戶端進行的任何呼叫產生子區段。

With OpenTelemetry SDK

針對下列程式碼範例，您將需要下列相依性：

```
pip install opentelemetry-instrumentation-botocore
```

以程式設計方式使用 OpenTelemetry Botocore 檢測來檢測應用程式中的所有 Boto3 用戶端。下列範例示範 `botocore` 檢測。

```
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}")
```

透過請求檢測傳出的 HTTP 呼叫

With X-Ray SDK

X-Ray Python SDK 透過修補請求程式庫，透過請求追蹤傳出的 HTTP 呼叫。如需詳細資訊，請參閱使用適用於 Python 的 X-Ray 開發套件追蹤對下游 HTTP Web 服務的呼叫。在您的應用程式中，您可以使用 `patch_all()`方法來檢測所有程式庫，或使用 選擇性地修補請求程式庫`patch(['requests']))`。任何選項都會檢測`requests`程式庫，為透過 進行的任何呼叫產生子區段`requests`。

With OpenTelemetry SDK

針對下列程式碼範例，您將需要下列相依性：

```
pip install opentelemetry-instrumentation-requests
```

以程式設計方式使用 OpenTelemetry Requests Instrumentation 來檢測請求程式庫，為應用程式中提出的 HTTP 請求產生追蹤。如需詳細資訊，請參閱 OpenTelemetry 請求檢測 。下列範例示範程式`requests`庫檢測。

```
from opentelemetry.instrumentation.requests import RequestsInstrumentor

# Instrument Requests
RequestsInstrumentor().instrument()

...

    example_session = requests.Session()
    example_session.get(url="https://example.com")
```

或者，您也可以檢測基礎`urllib3`程式庫來追蹤 HTTP 請求：

```
# 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")
```

其他程式庫的檢測支援

您可以在支援的程式庫、架構、應用程式伺服器和 JVM 下找到 OpenTelemetry Python 支援的程式庫檢測的完整清單。 JVMs

或者，您可以搜尋 OpenTelemetry 登錄檔，了解 OpenTelemetry 是否支援檢測。請參閱 登錄檔以開始搜尋。

手動建立追蹤資料

您可以在 `xray_recorder` Python 應用程式中使用 建立客群和子客群。如需詳細資訊，請參閱手動檢測 Python 程式碼。您也可以手動將註釋和中繼資料新增至追蹤資料。

使用 OpenTelemetry SDK 建立範圍

使用 `start_as_current_span` API 啟動跨度並將其設定為建立跨度。如需建立跨度的範例，請參閱建立跨度。一旦跨度開始並在目前範圍內，您可以透過新增屬性、事件、例外狀況、連結等來新增更多資訊。如同我們在 X-Ray 中擁有區段和子區段的方式，OpenTelemetry 中有不同類型的跨度。只有SERVER種類範圍會轉換為 X-Ray 區段，其他則轉換為 X-Ray 子區段。

```
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 將註釋和中繼資料新增至追蹤

X-Ray Python SDK 提供不同的 APIs，put_annotation以及put_metadata將註釋和中繼資料新增至追蹤。在 OpenTelemetry SDK 中，註釋和中繼資料只是跨度上的屬性，透過 set_attribute API 新增。

您希望它們在追蹤上成為註釋的跨度屬性會新增到保留索引鍵下方aws.xray.annotations，其值是註釋的索引鍵/值對清單。所有其他跨度屬性會成為轉換區段或子區段上的中繼資料。

此外，如果您使用 ADOT 收集器，您可以透過在收集器組態indexed_attributes中指定 來設定哪些跨度屬性應該轉換為 X-Ray 註釋。

以下範例示範如何使用 OpenTelemetry SDK 將註釋和中繼資料新增至追蹤。

```
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 檢測

若要在 X-Ray 上監控 lambda 函數，您可以啟用 X-Ray，並將適當的許可新增至函數叫用角色。此外，如果您要從函數追蹤下游請求，則會使用 X-Ray Python SDK 檢測程式碼。

使用適用於 X-Ray 的 OpenTelemetry，建議在 Application Signals 關閉的情況下使用 CloudWatch Application Signals lambda 層。這會自動檢測您的函數，並產生函數調用的範圍和函數的任何下游請求。除了追蹤之外，如果您有興趣使用 Application Signals 來監控函數的運作狀態，請參閱在 Lambda 上啟用應用程式。

- 從適用於 AWS OpenTelemetry ARN 的 Lambda Layer 尋找函數所需的 Lambda layer ARNs，並將其新增。
- 為您的函數設定下列環境變數。
  - `AWS_LAMBDA_EXEC_WRAPPER=/opt/otel-instrument` – 這會載入函數的自動檢測
  - `OTEL_AWS_APPLICATION_SIGNALS_ENABLED=false` – 這會停用 Application Signals 監控

使用 Lambda 檢測手動建立跨度

此外，您可以在函數內產生自訂跨度，以追蹤工作。您可以只使用 `opentelemetry-api`套件搭配 Application Signals lambda layer 自動檢測來完成。

1. 在函數中包含 `opentelemetry-api`做為相依性
2. 下列程式碼片段是產生自訂範圍的範例

```
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
    }
```

遷移至 OpenTelemetry Ruby

若要將 Ruby 應用程式從 X-Ray SDK 遷移至 OpenTelemetry 檢測，請使用下列程式碼範例和手動檢測指引。

章節

使用 SDK 手動檢測您的解決方案

Tracing setup with X-Ray SDK

適用於 Ruby 的 X-Ray 開發套件需要您使用服務外掛程式來設定程式碼。

```
require 'aws-xray-sdk'

XRay.recorder.configure(plugins: [:ec2, :elastic_beanstalk])
```

Tracing setup with OpenTelemetry SDK

> **Note**
>
> X-Ray 遠端取樣目前無法針對 OpenTelemetry Ruby 進行設定。

對於 Ruby on Rails 應用程式，請將您的組態碼放在 Rails 初始化器中。如需詳細資訊，請參閱 入門。對於所有手動檢測的 Ruby 程式，您必須使用 `OpenTelemetry::SDK.configure`方法來設定 OpenTelemetry Ruby SDK。

首先，安裝下列套件：

```
bundle add opentelemetry-sdk opentelemetry-exporter-otlp opentelemetry-propagator-
xray
```

接著，透過程式初始化時執行的組態程式碼來設定 OpenTelemetry SDK。建議您設定下列元件：

- `OTLP Exporter` – 將追蹤匯出至 CloudWatch 代理程式和 OpenTelemetry 收集器時需要
- `An AWS X-Ray Propagator` – 將追蹤內容傳播至與 X-Ray 整合 AWS 的服務時需要

```
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也有程式庫檢測的概念。啟用這些功能會自動建立 AWS SDK 等程式庫的跨度。OpenTelemetry 提供啟用所有程式庫檢測的選項，或指定要啟用哪些程式庫檢測。

若要啟用所有檢測，請先安裝 opentelemetry-instrumentation-all套件：

```
bundle add opentelemetry-instrumentation-all
```

接著，更新組態以啟用所有程式庫檢測，如下所示：

```
require 'opentelemetry/instrumentation/all'
...

OpenTelemetry::SDK.configure do |c|
   ...

  c.use_all() # Enable all instrumentations
end
```

OpenTelemetry SDKs也有程式庫檢測的概念。啟用這些功能會自動建立 AWS SDK 等程式庫的跨度。OpenTelemetry 提供啟用所有程式庫檢測的選項，或指定要啟用哪些程式庫檢測。

若要啟用所有檢測，請先安裝 opentelemetry-instrumentation-all套件：

```
bundle add opentelemetry-instrumentation-all
```

接著，更新組態以啟用所有程式庫檢測，如下所示：

```
require 'opentelemetry/instrumentation/all'
...

OpenTelemetry::SDK.configure do |c|
   ...

  c.use_all() # Enable all instrumentations
end
```

追蹤傳入的請求 (Rails 檢測)

With X-Ray SDK

使用 X-Ray 開發套件時，會在初始化時為 Rails 架構設定 X-Ray 追蹤。

範例 – 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，安裝下列套件：

```
bundle add opentelemetry-instrumentation-rack opentelemetry-instrumentation-
rails opentelemetry-instrumentation-action_pack opentelemetry-instrumentation-
active_record opentelemetry-instrumentation-action_view
```

接著，更新組態以啟用 Rails 應用程式的檢測，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 SDK 檢測

With X-Ray SDK

若要檢測來自 AWS SDK 的傳出 AWS 請求， AWS 開發套件用戶端會使用 X-Ray 進行修補，如下列範例所示：

```
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 適用於 Ruby V3 的 SDK 支援記錄和發出 OpenTelemetry 追蹤。如需有關如何為服務用戶端設定 OpenTelemetry 的資訊，請參閱《適用於 Ruby 的 AWS SDK》中的設定可觀測性功能。

檢測傳出的 HTTP 呼叫

對外部服務進行 HTTP 呼叫時，如果自動檢測不可用或無法提供足夠的詳細資訊，您可能需要手動檢測呼叫。

With X-Ray SDK

為了檢測下游呼叫，適用於 Ruby 的 X-Ray 開發套件用於修補應用程式使用的`net/http`程式庫：

```
require 'aws-xray-sdk'

config = {
  name: 'my app',
  patch: %I[net_http]
}

XRay.recorder.configure(config)
```

With OpenTelemetry SDK

若要使用 OpenTelemetry 啟用net/http檢測，請先安裝 opentelemetry-instrumentation-net_http套件：

```
bundle add opentelemetry-instrumentation-net_http
```

接著，更新組態以啟用net/http檢測，如下所示：

```
OpenTelemetry::SDK.configure do |c|
   ...

  c.use 'OpenTelemetry::Instrumentation::Net::HTTP'
  ...

end
```

其他程式庫的檢測支援

您可以在 opentelemetry-ruby-contrib 下找到 OpenTelemetry Ruby 支援的程式庫檢測的完整清單。

或者，您可以搜尋 OpenTelemetry 登錄檔，了解 OpenTelemetry 是否支援檢測。如需詳細資訊，請參閱 登錄檔。

手動建立追蹤資料

With X-Ray SDK

使用 X-Ray，aws-xray-sdk套件會要求您手動建立區段及其子區段，以追蹤您的應用程式。您也可以將 X-Ray 註釋和中繼資料新增至您的區段或子區段：

```
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

您可以使用自訂範圍來監控檢測程式庫未擷取的內部活動效能。請注意，只有種類的伺服器會轉換為 X-Ray 區段，所有其他範圍則會轉換為 X-Ray 子區段。根據預設，跨度為 INTERNAL。

首先，建立追蹤器以產生跨度，您可以透過 `OpenTelemetry.tracer_provider.tracer('<YOUR_TRACER_NAME>')`方法取得。這將提供在您應用程式的 OpenTelemetry 組態中全域註冊的 Tracer 執行個體。整個應用程式都有單一追蹤器是很常見的。建立 OpenTelemetry 追蹤器並使用它來建立跨度：

```
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 將註釋和中繼資料新增至追蹤

使用 `set_attribute`方法將屬性新增至每個範圍。請注意，根據預設，所有這些跨度屬性都會轉換為 X-Ray 原始資料中的中繼資料。若要確保屬性轉換為註釋而非中繼資料，您可以將該屬性索引鍵新增至`aws.xray.annotations`屬性清單。如需詳細資訊，請參閱啟用自訂 X-Ray 註釋。

```
# 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 手動檢測

With X-Ray SDK

在 Lambda 上啟用主動追蹤之後，使用 X-Ray SDK 不需要其他組態。Lambda 將建立代表 Lambda 處理常式調用的區段，而且您可以使用 X-Ray SDK 建立子區段或檢測程式庫，而不需要任何額外的組態。

With OpenTelemetry SDK

考慮以下範例 Lambda 函數程式碼（無需檢測）：

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

若要手動檢測 Lambda，您需要：

1. 為您的 Lambda 新增下列 Gem

   ```
   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 Handler 外部初始化 OpenTelemetry SDK。建議使用下列設定 OpenTelemetry SDK：

   1. 具有 X-Ray UDP 跨度匯出器的簡易跨度處理器，可將追蹤傳送至 Lambda 的 UDP X-Ray 端點
   2. X-Ray Lambda 傳播器
   3. `service_name` 要設定為 Lambda 函數名稱的組態

3. 在 Lambda 處理常式類別中，新增下列行來檢測 Lambda 處理常式：

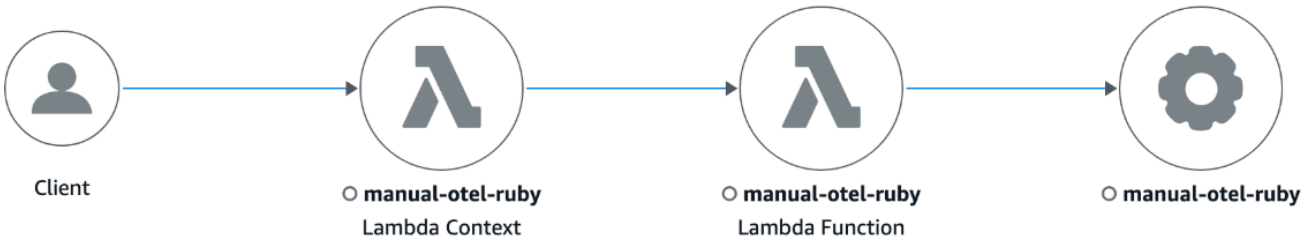
```
  class Handler
        extend OpenTelemetry::Instrumentation::AwsLambda::Wrap
        ...

        instrument_handler :process
    end
```

下列程式碼示範必要的變更之後的 Lambda 函數。您可以建立額外的自訂範圍，以補充自動提供的範圍。

```
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
```

以下是以 Ruby 編寫之經檢測 Lambda 函數的範例追蹤映射。



您也可以使用 Lambda 層來設定 Lambda 的 OpenTelemetry。如需詳細資訊，請參閱 OpenTelemetry AWS-Lambda 檢測。

使用 建立 X-Ray 資源 AWS CloudFormation

AWS X-Ray 已與 整合 AWS CloudFormation，這項服務可協助您建立和設定 AWS 資源的模型，以減少建立和管理資源和基礎設施的時間。您可以建立範本來描述您想要的所有 AWS 資源，並為您 AWS CloudFormation 佈建和設定這些資源。

使用 時 AWS CloudFormation，您可以重複使用範本，以一致且重複地設定 X-Ray 資源。描述您的資源一次，然後在多個 AWS 帳戶 和 區域中逐一佈建相同的資源。

X-Ray 和 AWS CloudFormation 範本

若要佈建和設定 X-Ray 和相關服務的資源，您必須了解 AWS CloudFormation 範本。範本是以 JSON 或 YAML 格式化的文本檔案。這些範本說明您要在 AWS CloudFormation 堆疊中佈建的資源。如果您不熟悉 JSON 或 YAML，您可以使用 AWS CloudFormation 設計工具來協助您開始使用 AWS CloudFormation 範本。如需更多詳細資訊，請參閱 AWS CloudFormation 使用者指南 中的 什麼是 AWS CloudFormation 設計器？。

X-Ray 支援在其中建立 `AWS::XRay::SamplingRule`、 `AWS::XRay::Group`和 `AWS::XRay::ResourcePolicy` 資源 AWS CloudFormation。如需詳細資訊，包括 JSON 和 YAML 範本的範例，請參閱AWS CloudFormation 《 使用者指南》中的 X-Ray 資源類型參考。

進一步了解 AWS CloudFormation

若要進一步了解 AWS CloudFormation，請參閱下列資源：

- AWS CloudFormation
- AWS CloudFormation 使用者指南
- AWS CloudFormation API 參考
- AWS CloudFormation 命令列界面使用者指南

標記 X-Ray 取樣規則和群組

標籤是您可以用來識別和組織 AWS 資源的單字或片語。您可以將多個標籤新增至每個資源。每個標籤都包含您定義的索引鍵和選用值。例如，標籤索引鍵可能是 **`domain`**，而標籤值可能是 **`example.com`**。您可以根據您新增的標籤來搜尋和篩選資源。如需如何使用標籤的詳細資訊，請參閱《 AWS 一般參考》中的標記 AWS 資源。

您可以使用標籤來強制執行 CloudFront 分佈上的標籤型許可。如需詳細資訊，請參閱使用資源標籤控制對 資源的存取 AWS。

> **Note**
>
> 標籤編輯器和資源AWS 群組目前不支援 X-Ray 資源。您可以使用 AWS X-Ray 主控台或 API 新增和管理標籤。

您可以使用 X-Ray 主控台、API、 AWS CLI SDKs 和 將標籤套用至資源 AWS Tools for Windows PowerShell。如需詳細資訊，請參閱下列 文件：

- X-Ray API – 請參閱 AWS X-Ray API 參考中的下列操作：
  - ListTagsForResource
  - CreateSamplingRule
  - CreateGroup
  - TagResource
  - UntagResource
- AWS CLI – 請參閱《 AWS CLI 命令參考》中的 Xray
- 開發套件 – 請參閱 AWS 說明文件頁面上適用的開發套件說明文件

> **Note**
>
> 如果您無法在 X-Ray 資源上新增或變更標籤，或無法新增具有特定標籤的資源，您可能沒有執行此操作的許可。若要請求存取，請聯絡您企業中具有 X-Ray 管理員許可 AWS 的使用者。

主題

標籤限制

下列限制適用於標籤。

- 每一資源最多標籤數：50
- 金鑰長度上限 - 128 個 Unicode 字元
- 數值長度上限 - 256 個 Unicode 字元
- 金鑰與值的有效值 – a-z、A-Z、0-9、空格和下列字元：_ . : / = + - 及 @
- 標籤鍵與值皆區分大小寫。
- 請不要使用 `aws:` 做為金鑰的字首；它保留供 AWS 使用。

> **Note**
>
> 您無法編輯或刪除系統標籤。

在 主控台中管理標籤

您可以在建立 X-Ray 群組或取樣規則時新增選用標籤。稍後也可以在主控台中變更或刪除標籤。

下列程序說明如何在 X-Ray 主控台中新增、編輯和刪除群組的標籤和取樣規則。

主題

將標籤新增至新群組（主控台）

當您建立新的 X-Ray 群組時，您可以在建立群組頁面上新增選用標籤。

1. 登入 AWS Management Console ，並在 https：//https://console.aws.amazon.com/xray/home 開啟 X-Ray 主控台。
2. 在導覽窗格中，展開組態，然後選擇群組。
3. 選擇建立群組。
4. 在建立群組頁面上，指定群組的名稱和篩選條件表達式。如需這些屬性的相關資訊，請參閱設定群組。
5. 在標籤中，輸入標籤索引鍵，並選擇性地輸入標籤值。例如，您可以輸入 的標籤索引鍵**Stage**，以及 的標籤值**Production**，以表示此群組是供生產使用。當您新增標籤時，會視需要顯示新行，供您新增另一個標籤。如需標籤的限制，請參閱本主題標籤限制中的 。
6. 新增標籤完成後，請選擇建立群組。

將標籤新增至新的取樣規則（主控台）

當您建立新的 X-Ray 取樣規則時，您可以在建立取樣規則頁面上新增標籤。

1. 登入 AWS Management Console ，並在 https：//https://console.aws.amazon.com/xray/home 開啟 X-Ray 主控台。
2. 在導覽窗格中，展開組態，然後選擇取樣。
3. 選擇建立取樣規則。
4. 在建立取樣規則頁面上，指定名稱、優先順序、限制、相符條件和相符屬性。如需這些屬性的相關資訊，請參閱設定 取樣規則。
5. 在標籤中，輸入標籤索引鍵，並選擇性地輸入標籤值。例如，您可以輸入 的標籤索引鍵**Stage**，以及 的標籤值**Production**，以表示此取樣規則是供生產使用。當您新增標籤時，會視需要顯示新行，供您新增另一個標籤。如需標籤的限制，請參閱本主題標籤限制中的 。
6. 新增標籤完成後，請選擇建立取樣規則。

編輯或刪除群組的標籤（主控台）

您可以在編輯群組頁面上變更或刪除 X-Ray 群組上的標籤。

1. 登入 AWS Management Console ，並在 https：//https://console.aws.amazon.com/xray/home. 開啟 X-Ray 主控台。
2. 在導覽窗格中，展開組態，然後選擇群組。
3. 在群組表格中，選擇群組的名稱。
4. 在編輯群組頁面的標籤中，編輯標籤索引鍵和值。您無法有重複的標籤索引鍵。標籤值為選用；您可以視需要刪除值。如需編輯群組頁面上其他屬性的詳細資訊，請參閱 設定 群組。如需標籤的限制，請參閱本主題標籤限制中的 。
5. 若要刪除標籤，請選擇標籤右側的 X。
6. 當您完成編輯或刪除標籤時，請選擇更新群組。

編輯或刪除取樣規則的標籤 （ 主控台 ）

您可以在編輯取樣規則頁面上變更或刪除 X-Ray 取樣規則上的標籤。

1. 登入 AWS Management Console ，並在 https：//https://console.aws.amazon.com/xray/home 開啟 X-Ray 主控台。
2. 在導覽窗格中，展開組態，然後選擇取樣。
3. 在取樣規則表格中，選擇取樣規則的名稱。
4. 在標籤中，編輯標籤索引鍵和值。您無法有重複的標籤索引鍵。標籤值為選用；您可以視需要刪除值。如需編輯取樣規則頁面上其他屬性的詳細資訊，請參閱 設定 取樣規則。如需標籤的限制，請參閱本主題標籤限制中的 。
5. 若要刪除標籤，請選擇標籤右側的 X。
6. 當您完成編輯或刪除標籤時，請選擇更新取樣規則。

在 中管理標籤 AWS CLI

您可以在建立 X-Ray 群組或取樣規則時新增標籤。您也可以使用 AWS CLI 來建立和管理標籤。若要更新現有群組或取樣規則上的標籤，請使用 AWS X-Ray 主控台或 TagResource 或 UntagResource APIs。

主題

- 將標籤新增至新的 X-Ray 群組或取樣規則 (CLI)
- 將標籤新增至現有資源 (CLI)
- 列出資源上的標籤 (CLI)

- 刪除資源上的標籤 (CLI)

將標籤新增至新的 X-Ray 群組或取樣規則 (CLI)

若要在建立新的 X-Ray 群組或取樣規則時新增選用標籤，請使用下列其中一個命令。

- 若要將標籤新增至新群組，請執行下列命令，將 *group_name* 取代為您的群組名稱、將 *mydomain.com* 取代為您的服務端點、將 *key_name* 取代為標籤索引鍵，以及選擇性地將#取代為標籤值。如需如何建立群組的詳細資訊，請參閱 群組。

```
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"}]
```

  以下是範例。

```
aws xray create-group \
   --group-name "AdminGroup" \
   --filter-expression "service(\"mydomain.com\") {fault OR error}" \
   --tags [{"Key": "Stage","Value": "Prod"},{"Key": "Department","Value": "QA"}]
```

- 若要將標籤新增至新的取樣規則，請執行下列命令，將 *key_name* 取代為標籤索引鍵，並選擇性地將#取代為標籤值。此命令會將 `--sampling-rule` 參數中的值指定為 JSON 檔案。如需如何建立取樣規則的詳細資訊，請參閱 抽樣規則。

```
aws xray create-sampling-rule \
   --cli-input-json file://file_name.json
```

  以下是 `--cli-input-json` 參數所指定之 JSON *file_name.json* 的內容。

```
{
    "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"
           }
          ]
}
```

下列是範例命令。

```
aws xray create-sampling-rule \
   --cli-input-json file://9000-base-scorekeep.json
```

以下是 --cli-input-json 參數指定的範例9000-base-scorekeep.json檔案內容。

```
{
    "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"
            }
           ]
}
```

將標籤新增至現有資源 (CLI)

您可以執行 `tag-resource`命令，將標籤新增至現有的 X-Ray 群組或取樣規則。此方法可能比執行 `update-group`或 新增標籤更為簡單`update-sampling-rule`。

若要將標籤新增至群組或取樣規則，請執行下列命令，以資源的 ARN 取代 ARN，並指定您要新增的標籤索引鍵和選用值。

```
aws xray tag-resource \
   --resource-arn "ARN" \
   --tag-keys [{"Key":"key_name","Value":"value"}, {"Key":"key_name","Value":"value"}]
```

以下是範例。

```
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"}]
```

列出資源上的標籤 (CLI)

您可以執行 `list-tags-for-resource`命令來列出 X-Ray 群組或取樣規則的標籤。

若要列出與群組或取樣規則相關聯的標籤，請執行下列命令，以資源的 ARN 取代 ARN。

```
aws xray list-tags-for-resource \
   --resource-arn "ARN"
```

以下是範例。

```
aws xray list-tags-for-resource \
```

```
   --resource-arn "arn:aws:xray:us-east-2:01234567890:group/AdminGroup"
```

刪除資源上的標籤 (CLI)

您可以執行 `untag-resource`命令，從 X-Ray 群組或取樣規則中移除標籤。

若要從群組或抽樣規則移除標籤，請執行下列命令，將 ARN 取代為資源的 ARN，並指定您要移除的標籤索引鍵。

您只能使用 `untag-resource`命令移除整個標籤。若要移除標籤值，請使用 X-Ray 主控台，或刪除標籤，並新增具有相同索引鍵但不同或空白值的新標籤。

```
aws xray untag-resource \
   --resource-arn "ARN" \
   --tag-keys ["key_name","key_name"]
```

以下是範例。

```
aws xray untag-resource \
   --resource-arn "arn:aws:xray:us-east-2:01234567890:group/group_name" \
   --tag-keys ["Stage","Department"]
```

根據標籤控制對 X-Ray 資源的存取

您可以將標籤連接至 X-Ray 群組或取樣規則，或在請求中將標籤傳遞至 X-Ray。如需根據標籤控制存取，請使用 `xray:ResourceTag/key-name`、`aws:RequestTag/key-name` 或 `aws:TagKeys` 條件索引鍵，在政策的條件元素中，提供標籤資訊。若要進一步了解這些條件索引鍵，請參閱使用 AWS 資源標籤控制對資源的存取。

若要檢視身分型政策範例，以根據該資源上的標籤來限制存取資源，請參閱根據標籤管理對 X-Ray 群組和抽樣規則的存取。

故障診斷 AWS X-Ray

本主題列出使用 X-Ray API、主控台或 SDKs常見錯誤和問題。如果您發現未列在此處的問題，您可以使用此頁面上的 Feedback (意見回饋) 按鈕來報告。

章節

X-Ray 追蹤映射和追蹤詳細資訊頁面

如果您在使用 X-Ray 追蹤映射和追蹤詳細資訊頁面時遇到問題，下列各節可協助您：

我沒有看到所有 CloudWatch 日誌

如何設定日誌，使其顯示在 X-Ray 追蹤映射和追蹤詳細資訊頁面中，取決於 服務。

- API Gateway 日誌只會在 API Gateway 中開啟記錄日誌時出現。

並非所有服務映射節點都支援檢視相關聯的日誌。檢視下列節點類型的日誌：

- Lambda 內容
- Lambda 函數
- API Gateway 階段
- Amazon ECS 叢集
- Amazon ECS 執行個體
- Amazon ECS 服務
- Amazon ECS 任務
- Amazon EKS 叢集
- Amazon EKS 命名空間
- Amazon EKS 節點

- Amazon EKS Pod
- Amazon EKS 服務

我在 X-Ray 追蹤地圖上看不到所有警示

如果與該節點相關聯的任何警示處於 ALARM 狀態，X-Ray 追蹤映射只會顯示節點的警示圖示。

追蹤映射使用以下邏輯將警示與節點建立關聯：

- 如果節點代表 AWS 服務，則具有與該服務相關聯之命名空間的所有警示都會與該節點相關聯。例如，`AWS::Kinesis` 類型的節點會與所有根據 CloudWatch 命名空間 `AWS/Kinesis` 中指標為基礎的警示進行連結。
- 如果節點代表 AWS 資源，則會連結該特定資源上的警示。例如，`AWS::DynamoDB::Table` 類型且名稱為 “MyTable” 的節點會與所有以命名空間為 `AWS/DynamoDB`，且已將 `TableName` 維度設為 `MyTable` 指標為基礎的警示連結。
- 如果節點是未知類型 (由名稱周圍的虛線外框表示)，則不會有任何警示與該節點建立關聯。

我在追蹤地圖上看不到一些 AWS 資源

並非每個 AWS 資源都由專用節點表示。有些 AWS 服務由單一節點表示，用於所有對服務的請求。下列資源類型會以每個資源一個節點的方式顯示：

- `AWS::DynamoDB::Table`
- `AWS::Lambda::Function`

  Lambda 函數由兩個節點表示，一個用於 Lambda 容器，另一個用於函數。這有助於識別 Lambda 函數的冷啟動問題。Lambda 容器節點與警示和儀表板建立關聯的方式同 Lambda 函數節點一樣。
- `AWS::ApiGateway::Stage`
- `AWS::SQS::Queue`
- `AWS::SNS::Topic`

追蹤映射上有太多節點

請使用 X-Ray 群組，以將您的映射分成多個映射。如需詳細資訊，請參閱[搭配群組使用篩選條件表達式](#)。

適用於 Java 的 X-Ray 開發套件

錯誤：Exception in thread "Thread-1" com.amazonaws.xray.exceptions.SegmentNotFoundException: Failed to begin subsegment named 'AmazonSNS': segment cannot be found. (執行緒 "Thread-1" 中發生異常 com.amazonaws.xray.exceptions.SegmentNotFoundException：無法開始名為 'AmazonSNS' 的子區段：找不到子區段。)

此錯誤表示 X-Ray 開發套件嘗試記錄傳出呼叫 AWS，但找不到開啟的區段。發生此問題的可能情況如下：

- 未設定 servlet 篩選條件 – X-Ray SDK 會使用名為 的篩選條件為傳入請求建立區段`AWSXRayServletFilter`。設定 servlet 篩選條件來檢測傳入請求。
- 您正在 servlet 程式碼之外使用經檢測的用戶端 – 如果您使用經檢測的用戶端在啟動程式碼或其他未執行的程式碼中呼叫 以回應傳入的請求，則必須手動建立客群。如需範例，請參閱 檢測啟動程式碼。
- 您在工作者執行緒中使用經檢測的用戶端 – 當您建立新的執行緒時，X-Ray 記錄器會失去其對開啟區段的參考。您可以使用 `getTraceEntity` 和 `setTraceEntity` 方法來取得目前區段或子區段的參考 (`Entity`)，並將其傳遞給執行緒內部的記錄器。如需範例，請參閱在工作者執行緒中使用受檢測用戶端。

適用於 Node.js 的 X-Ray 開發套件

問題：「CLS 並未使用 Sequelize」

使用 `cls`方法將適用於 Node.js 的 X-Ray 開發套件命名空間傳遞至 Sequelize。

```
var AWSXRay = require('aws-xray-sdk');
const Sequelize = require('sequelize');
Sequelize.cls = AWSXRay.getNamespace();
const sequelize = new Sequelize(...);
```

問題：「CLS 並未使用 Bluebird」

使用 `cls-bluebird` 來使 Bluebird 使用 CLS。

```
var AWSXRay = require('aws-xray-sdk');
var Promise = require('bluebird');
var clsBluebird = require('cls-bluebird');
```

```
clsBluebird(AWSXRay.getNamespace());
```

X-Ray 協助程式

問題：「精靈使用錯誤的登入資料」

協助程式使用 AWS SDK 載入登入資料。若您使用多個方法提供登入資料，便會使用優先順序最高的方法。如需詳細資訊，請參閱「執行精靈」。

的文件歷史記錄 AWS X-Ray

下表說明 文件的重要變更 AWS X-Ray。如需有關此文件更新的通知，您可以訂閱 RSS 訂閱源。

文件最近更新時間：2024 年 3 月 7 日

變更	描述	日期
已新增的功能	從 X-Ray 遷移至 OpenTelemetry。如需詳細資訊，請參閱從 X-Ray 檢測遷移至 OpenTelemetry 檢測。	2025 年 6 月 13 日
已新增的功能	AWS X-Ray 現在支援交易搜尋。如需詳細資訊，請參閱交易搜尋。	2024 年 11 月 21 日
已新增的功能	AWS X-Ray 現在支援 OpenTelemetry Protocol (OTLP) 端點。如需詳細資訊，請參閱 OpenTelemetry。	2024 年 11 月 21 日
已新增的功能	X-Ray 現在會記錄資料事件，包括 `PutTraceSegments` 、`GetTraceSummaries` 和 `BatchGetTraces` 目標 AWS CloudTrail。X-Ray 現在也會將`GetSamplingStatisticSummaries` 管理事件記錄到 CloudTrail。如需詳細資訊，請參閱使用 記錄 X-Ray API 呼叫 AWS CloudTrail。	2024 年 3 月 7 日
已新增的功能	X-Ray 現在支援透過 OpenTelemetry 或任何其他符合 W3C 追蹤內容規格的架構建立的追蹤 IDs。如需詳細資	2023 年 10 月 25 日

	訊，請參閱將追蹤資料傳送至 X-Ray。	
已新增的功能	您現在可以設定 Amazon SNS 主動追蹤，讓您在請求通過 Amazon SNS 主題時追蹤和分析請求。如需詳細資訊，請參閱 Amazon SNS 和 AWS X-Ray 。	2023 年 2 月 8 日
已更新適用於 Node.js 的 X-Ray 開發套件主題	新增使用 適用於 JavaScript 的 AWS SDK V3 檢測用戶端的詳細資訊。如需詳細資訊，請參閱使用適用於 Node.js 的 X-Ray AWS 開發套件追蹤 SDK 呼叫。	2023 年 2 月 7 日
更新 IAM 受管政策詳細資訊	已將跨帳戶可觀測性的 IAM 許可新增至 `AWSXRayReadOnlyAccess` 和 `AWSXRayFullAccess` `AWSXrayCrossAccountSharingConfiguration` 受管政策。如需詳細資訊，請參閱適用於 X-Ray 的 IAM 受管政策。	2023 年 2 月 7 日
已新增的功能	AWS X-Ray 現在支援跨帳戶可觀測性，可讓您監控和疑難排解跨 內多個帳戶的應用程式 AWS 區域。如需詳細資訊，請參閱跨帳戶追蹤。	2022 年 11 月 27 日

已新增的功能	您現在可以檢視訊息生產者、Amazon SQS 佇列和消費者之間的連結追蹤，提供從事件驅動應用程式傳送的追蹤連線檢視。如需詳細資訊，請參閱追蹤事件驅動的應用程式。	2022 年 11 月 20 日
更新 IAM 受管政策詳細資訊	新增將資源政策列出至 `AWSXRayReadOnlyAccess` 受管政策的 IAM 許可。如需詳細資訊，請參閱適用於 X-Ray 的 IAM 受管政策。	2022 年 11 月 15 日
已更新 IAM 主控台許可和受管政策詳細資訊	已更新 X-Ray 主控台使用的一組 IAM 許可，以及`AWSXRayReadOnlyAccess` 受管政策的描述。如需詳細資訊，請參閱使用 X-Ray 主控台。	2022 年 11 月 11 日
新增 AWS Distro for OpenTelemetry Ruby	AWS Distro for OpenTelemetry (ADOT) 提供一組單一開放原始碼 APIs、程式庫和代理程式，以收集分散式追蹤和指標。ADOT Ruby 可讓您檢測 Ruby 應用程式是否有 X-Ray 和其他追蹤後端。如需詳細資訊，請參閱 AWS Distro for OpenTelemetry Ruby。	2022 年 2 月 7 日
已新增的功能	您現在可以從 CloudWatch 主控台檢視追蹤並設定 X-Ray。如需詳細資訊，請參閱 X-Ray 主控台。	2022 年 1 月 24 日

整合式 CloudWatch RUM	使用 AWS X-Ray 和 CloudWatch RUM，您可以透過下游 AWS 受管服務來分析和偵錯從應用程式最終使用者開始的請求路徑。如需詳細資訊，請參閱 CloudWatch RUM 和 AWS X-Ray 。	2021 年 12 月 3 日
適用於 OpenTelemetry 的 Integrated AWS Distro	AWS Distro for OpenTelemetry (ADOT) 提供一組單一開放原始碼 APIs、程式庫和代理程式，用於收集分散式追蹤和指標。ADOT 可讓您檢測應用程式是否有 X-Ray 和其他追蹤後端。如需詳細資訊，請參閱檢測您的應用程式。	2021 年 9 月 23 日
已新增的功能	AWS X-Ray 現在與 Amazon Virtual Private Cloud 整合，可讓 Amazon VPC 中的資源與 X-Ray 服務通訊，而無需透過公有網際網路。如需詳細資訊，請參閱搭配使用 AWS X-Ray 與 VPC 端點。	2021 年 5 月 20 日
已新增的功能	AWS X-Ray 現在與 整合 AWS CloudFormation，可讓您佈建和設定 X-Ray 資源。如需詳細資訊，請參閱使用 CloudFormation 建立 X-Ray 資源。	2021 年 5 月 6 日

已新增的功能	AWS X-Ray 現在與 Amazon EventBridge 整合，以追蹤透過 EventBridge 傳遞的事件。這可提供使用者更完整的系統檢視。如需詳細資訊，請參閱 Amazon EventBridge 和 AWS X-Ray 。	2021 年 3 月 2 日
已將協助程式新增至 ECR	協助程式現在可以從 Amazon ECR 下載。如需詳細資訊，請參閱下載協助程式。	2021 年 3 月 1 日
已新增的功能	AWS X-Ray 現在支援 Amazon EventBridge 的洞察相關通知。這可讓您使用 EventBridge 對洞見採取自動動作。如需詳細資訊，請參閱 Insights Notifications。	2020 年 10 月 15 日
新增可下載的協助程式	AWS X-Ray 推出 Linux ARM64 的支援協助程式。如需詳細資訊，請參閱 AWS X-Ray daemonbrazil ws	2020 年 10 月 1 日
已新增的功能	AWS X-Ray 現在支援與 Amazon CloudWatch Synthetics 進行主動整合。這可讓您查看 Synthetics Canary 用戶端節點的詳細資訊，例如回應時間和狀態。您也可以根據來自 Synthetics Canary 用戶端節點的資訊，在 Analytics 主控台中進行分析。如需詳細資訊，請參閱使用 X-Ray 偵錯 CloudWatch 合成 Canary。	2020 年 9 月 24 日

已新增的功能	AWS X-Ray 現在支援追蹤 end-to-end工作流程 AWS Step Functions。您可以視覺化狀態機器的元件、識別效能瓶頸，以及對導致錯誤的請求進行故障診斷。如需詳細資訊，請參閱 AWS Step Functions 和 AWS X-Ray 。	2020 年 9 月 14 日
已新增的功能	AWS X-Ray 引入洞見，以持續分析帳戶中的追蹤資料，以識別應用程式中出現的問題。Insights 會記錄事件並追蹤事件影響，直到解決為止。如需詳細資訊，請參閱在 AWS X-Ray 主控台中使用洞見	2020 年 9 月 3 日
已新增的功能	AWS X-Ray 推出 Java 自動檢測代理程式，讓客戶無需修改現有的 Java 應用程式，即可收集追蹤資料。您現在可以追蹤以 Java Web 和 servlet 為基礎的應用程式，並將組態變更降至最低，無需變更程式碼。如需詳細資訊，請參閱適用於 AWS X-Ray Java 的自動檢測代理程式。	2020 年 9 月 3 日
已新增的功能	AWS X-Ray 已將新的群組頁面新增至 X-Ray 主控台，以協助簡化追蹤群組的建立和管理。如需詳細資訊，請參閱在 X-Ray 主控台中設定群組。	2020 年 8 月 24 日

已新增的功能	AWS X-Ray 現在可讓您將標籤新增至群組和取樣規則。您也可以根據標籤控制對群組和抽樣規則的存取。如需詳細資訊，請參閱標記 X-Ray 取樣規則和群組，以及根據標籤管理對 X-Ray 群組的存取和取樣規則。	2020 年 8 月 24 日

本文為英文版的機器翻譯版本，如內容有任何歧義或不一致之處，概以英文版為準。