aws

개발자 안내서

AWS X-Ray

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS X-Ray: 개발자 안내서

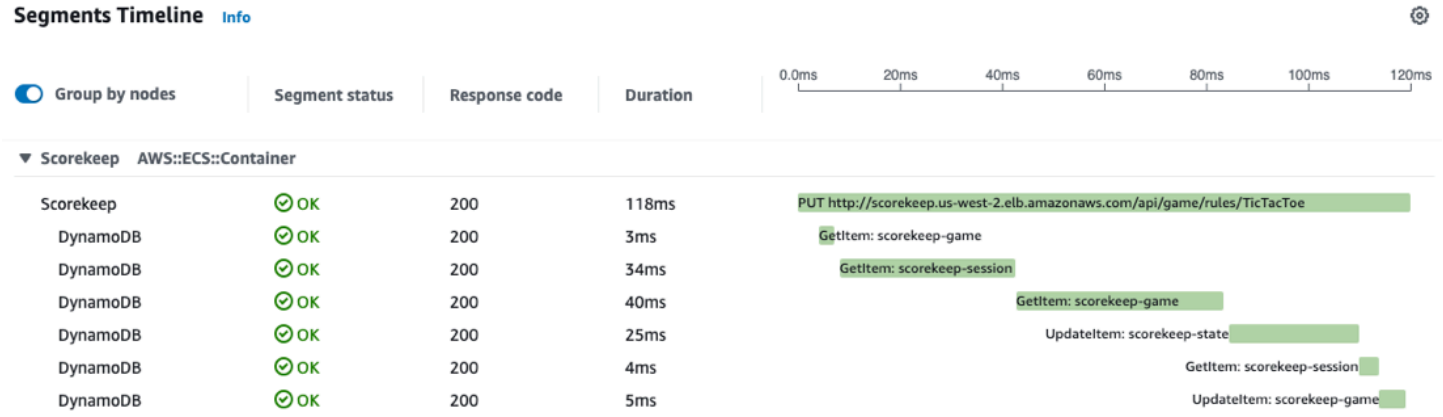
Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon의 상표 및 트레이드 드레스는 Amazon 외 제품 또는 서비스와 함께, Amazon 브랜드 이미지를 떨어뜨리거나 고객에게 혼동을 일으킬 수 있는 방식으로 사용할 수 없습니다. Amazon이 소유하지 않은 기타 모든 상표는 Amazon 계열사, 관련 업체 또는 Amazon의 지원 업체 여부에 상관없이 해당 소유자의 자산입니다.

Table of Contents

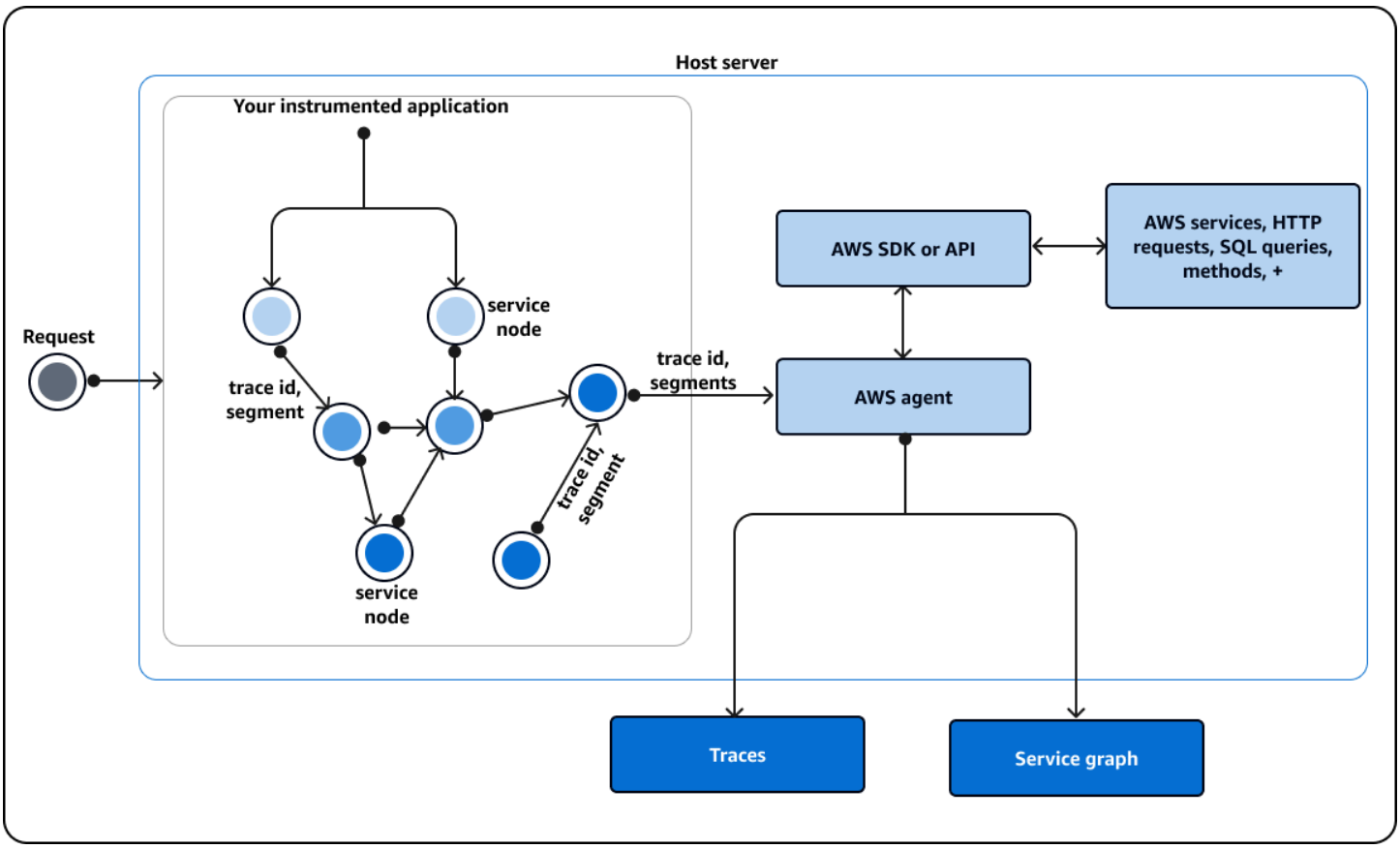
AWS X-Ray란 무엇인가요?

AWS X-Ray 는 애플리케이션이 처리하는 요청에 대한 데이터를 수집하고 해당 데이터를 보고, 필터링하고, 인사이트를 얻어 문제와 최적화 기회를 식별하는 데 사용할 수 있는 도구를 제공하는 서비스입니다. 애플리케이션에 대한 추적된 요청의 경우 요청 및 응답뿐만 아니라 애플리케이션이 다운스트림 AWS 리소스, 마이크로서비스, 데이터베이스 및 웹 APIs.



AWS X-Ray 는 X-Ray와 이미 통합된 애플리케이션 사용 외에도 애플리케이션에서 추적을 수신 AWS 서비스 합니다. 애플리케이션 계측에는 각 요청에 대한 메타데이터와 함께 애플리케이션 내의 수신 및 발신 요청 및 기타 이벤트에 대한 추적 데이터를 전송하는 작업이 포함됩니다. 많은 구성 시나리오에서는 구성 변경만 하면 됩니다. 예를 들어 Java 애플리케이션이 수행하는 모든 수신 HTTP 요청 및 다운스트림 호출을 계측할 수 AWS 서비스 있습니다. 애플리케이션의 엑스레이 트레이싱을 위한 계측에 사용할 수 있는 몇 가지 SDK, 에이전트 및 도구가 있습니다. 자세한 내용은 애플리케이션 계측하기를 참조하세요.

AWS 서비스 X-Ray와 통합된는 수신 요청에 추적 헤더를 추가하거나, 추적 데이터를 X-Ray로 보내거나, X-Ray 데몬을 실행할 수 있습니다. 예를 들어, AWS Lambda 는 Lambda 함수에 대한 요청에 대한 추적 데이터를 보내고 작업자에서 X-Ray 데몬을 실행하여 X-Ray SDK를 더 쉽게 사용할 수 있습니다.



트레이스 데이터를 직접 X-Ray로 전송하는 대신, 각 클라이언트 SDK는 UDP 트래픽을 수신 대기하는 대몬(daemon) 프로세스로 JSON 세그먼트 문서를 전송합니다. X-Ray 대몬(daemon)은 대기열에 세그먼트를 버퍼링하다가 일괄적으로 X-Ray로 업로드합니다. 데몬은 Linux, Windows 및 macOS에서 사용할 수 있으며 및 AWS Elastic Beanstalk AWS Lambda 플랫폼에 포함되어 있습니다.

X-Ray는 클라우드 애플리케이션을 지원하는 AWS 리소스의 트레이스 데이터를 사용하여 세부 트레이스 맵을 생성합니다. 트레이스 맵에는 클라이언트, 프런트엔드 서비스, 프런트엔드 서비스에서 요청을 처리하고 데이터를 유지하기 위해 직접적으로 호출하는 백엔드 서비스가 표시됩니다. 트레이스 맵을 사용하여 병목, 지연 시간 스파이크 등 애플리케이션의 성능을 개선하기 위해 해결할 수 있는 여러 문제를 식별할 수 있습니다.


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

X-Ray 시작하기

X-Ray를 사용하려면 다음 단계를 수행합니다.

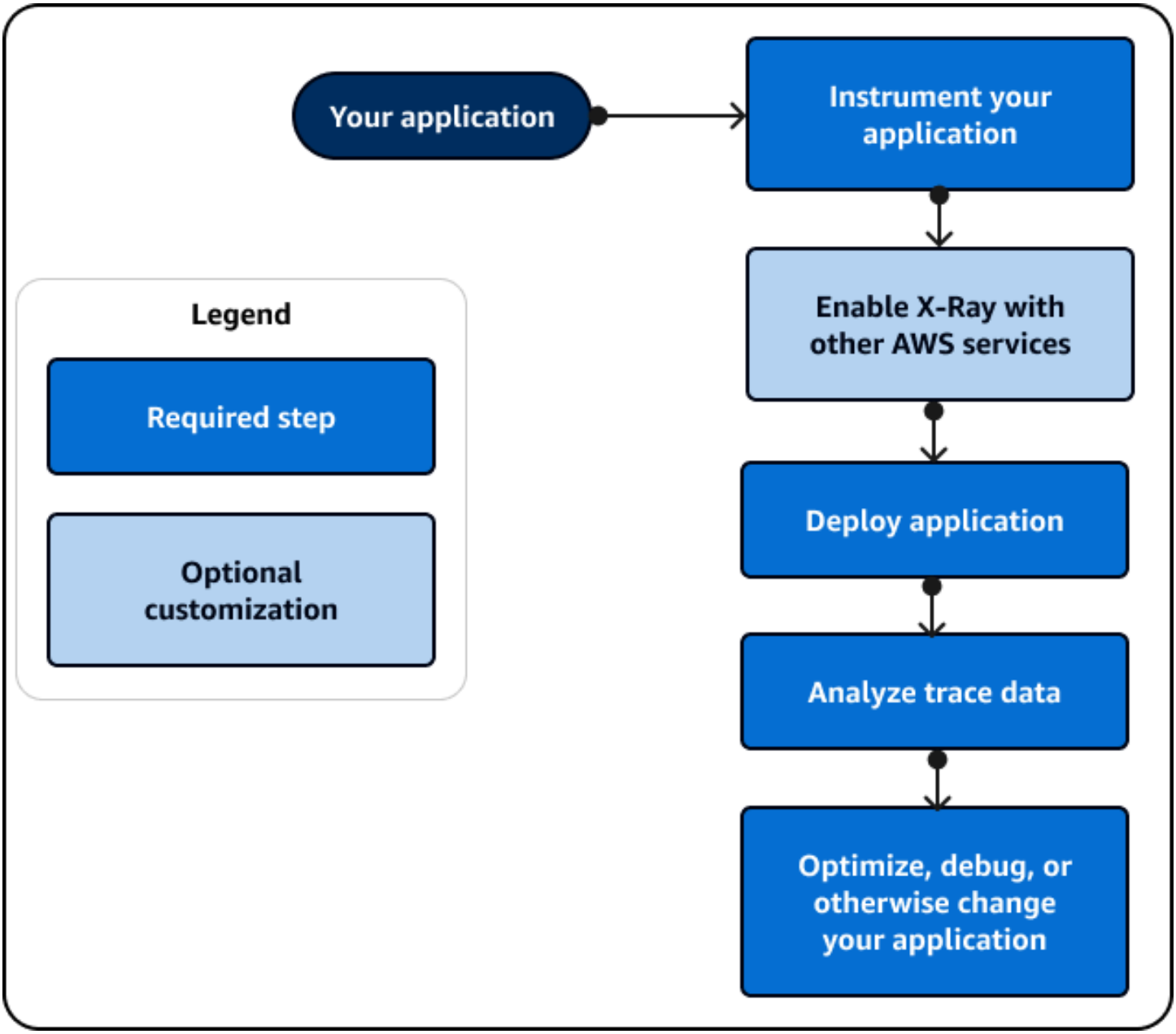
1. X-Ray에서 애플리케이션의 요청 처리 방식을 추적할 수 있도록 애플리케이션을 계측합니다.

   - X-Ray SDKs, X-Ray API, ADOT 또는 CloudWatch Application Signals를 사용하여 X-Ray로 트레이스 데이터를 전송합니다. 사용할 인터페이스에 대한 자세한 내용은 인터페이스 선택 단원을 참조하세요.

   계측에 대한 자세한 내용은 용 애플리케이션 계측 AWS X-Ray 단원을 참조하세요.
2. (선택 사항) X-Ray와 통합 AWS 서비스 되는 다른와 함께 작동하도록 X-Ray를 구성합니다. 트레이스를 샘플링하고 수신 요청에 헤더를 추가하고, 에이전트 또는 수집기를 실행하고, 트레이스 데이터를 X-Ray로 자동 전송할 수 있습니다. 자세한 내용은 다른 AWS X-Ray 와 통합 AWS 서비스 단원을 참조하십시오.
3. 계측된 애플리케이션을 배포합니다. 애플리케이션에서 요청을 수신하면 X-Ray SDK는 트레이스, 세그먼트 및 하위 세그먼트 데이터를 기록합니다. 이 단계에서는 IAM 정책을 설정하고 에이전트 또는 수집기를 배포해야 할 수도 있습니다.

   - AWS Distro for OpenTelemetry(ADOT) SDK 및 다양한 플랫폼에서 CloudWatch 에이전트를 사용하여 애플리케이션을 배포하는 스크립트 예제는 Application Signals 데모 스크립트를 참조하세요.
   - X-Ray SDK 및 X-Ray 대몬을 사용하여 애플리케이션을 배포하는 예제 스크립트는 AWS X-Ray 샘플 애플리케이션 단원을 참조하세요.
4. (선택 사항) 콘솔을 열어 데이터를 확인하고 분석합니다. 트레이스 맵, 서비스 맵 등의 GUI 표현을 통해 애플리케이션의 작동 방식을 검사할 수 있습니다. 콘솔의 그래픽 정보를 사용하여 애플리케이션을 최적화, 디버깅 및 이해할 수 있습니다. 콘솔 선택에 대한 자세한 내용은 콘솔 사용 단원을 참조하세요.

다음 다이어그램은 X-Ray 사용을 시작하는 방법을 보여줍니다.



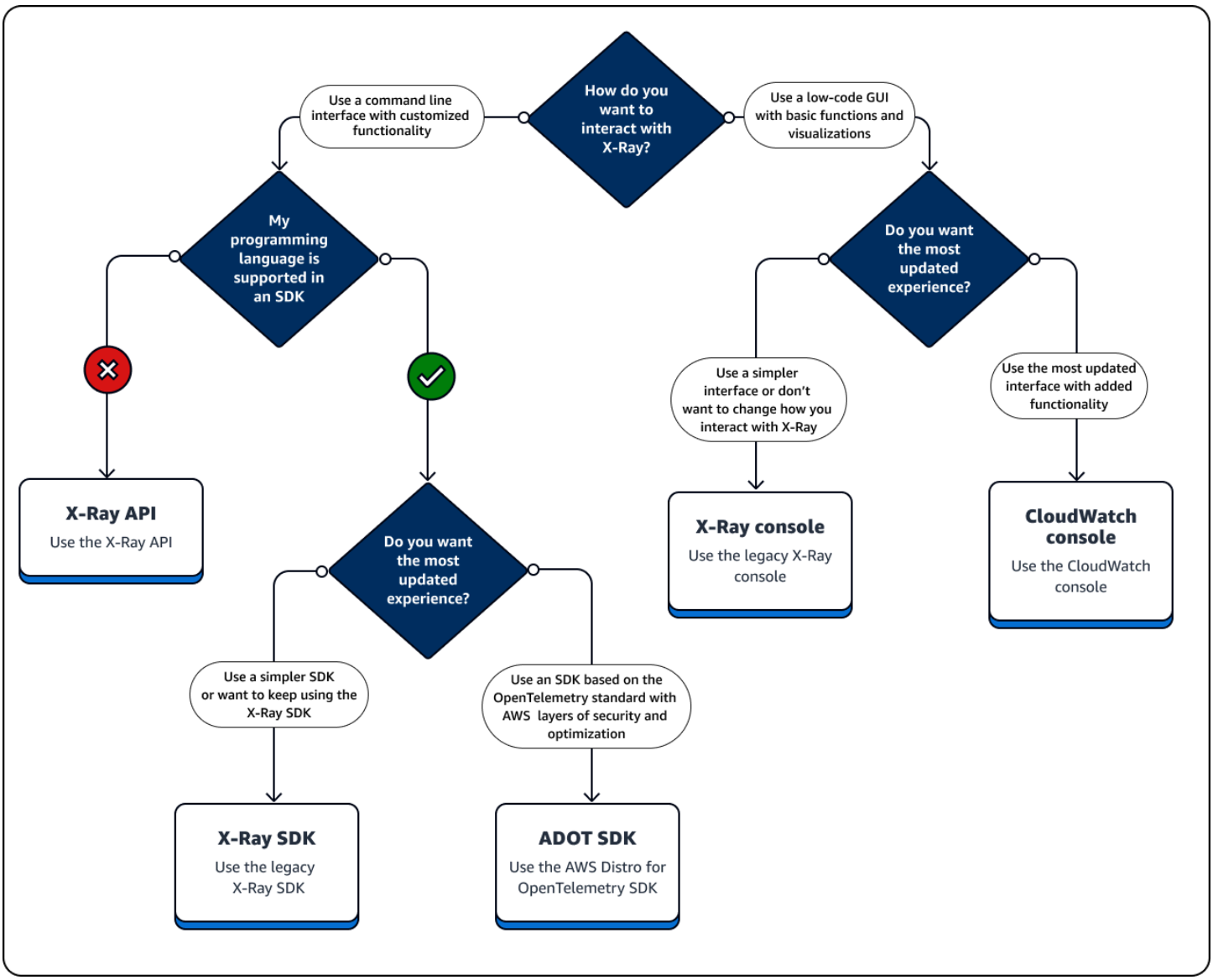
콘솔에서 사용할 수 있는 데이터 및 맵의 예를 보려면 트레이스 데이터를 생성할 수 있도록 이미 계측된 샘플 애플리케이션을 시작합니다. 몇 분 안에 트래픽을 생성하고, 세그먼트를 X-Ray로 전송하고, 트레이스 및 서비스 맵을 확인할 수 있습니다.

인터페이스 선택

AWS X-Ray 는 애플리케이션이 작동하는 방식과 애플리케이션이 다른 서비스 및 리소스와 얼마나 잘 상호 작용하는지에 대한 인사이트를 제공할 수 있습니다. 사용자가 애플리케이션을 계측하거나 구성하면 애플리케이션에서 요청을 처리할 때 X-Ray는 트레이스 데이터를 수집합니다. 이 트레이스 데이터를 분석하면 성능 문제를 파악하고 오류를 해결하고 리소스를 최적화할 수 있습니다. 이 안내서는 다음 지침에 따라 X-Ray와 상호 작용하는 방법을 보여줍니다.

- 빠르게 AWS Management Console 시작하려면를 사용하거나 사전 구축된 시각화를 사용하여 기본 작업을 수행할 수 있습니다.
  - X-Ray 콘솔의 모든 기능이 포함된 가장 최신의 사용자 환경을 원한다면 Amazon CloudWatch 콘솔을 선택합니다.
  - 더 간단한 인터페이스를 원하거나 X-Ray와 상호 작용하는 방식을 변경하고 싶지 않다면 X-Ray 콘솔을 사용합니다.
- 에서 제공할 AWS Management Console 수 있는 것보다 더 많은 사용자 지정 추적, 모니터링 또는 로깅 기능이 필요한 경우 SDK를 사용합니다.
  - 오픈 소스 OpenTelemetry SDK를 기반으로 하며 AWS 의 보안 및 최적화 계층이 추가된, 공급업체에 구애받지 않는 SDK를 원하는 경우 ADOT SDK를 선택합니다.
  - 더 간단한 SDK를 원하거나 애플리케이션 코드를 업데이트하지 않으려는 경우 X-Ray SDK를 선택합니다.
- SDK가 애플리케이션의 프로그래밍 언어를 지원하지 않는 경우 X-Ray API 작업을 사용합니다.

다음 다이어그램은 X-Ray와 상호 작용하는 방법을 선택하는 데 도움이 됩니다.



인터페이스 유형 살펴보기

- SDK 사용
- 콘솔 사용
- X-Ray API 사용

SDK 사용

명령줄 인터페이스를 사용하거나 AWS Management Console에서 사용할 수 있는 것보다 더 많은 사용자 지정 트레이스, 모니터링 또는 로깅 기능이 필요한 경우 SDK를 사용합니다. AWS SDK를 사용하

여 X-Ray APIs. AWS Distro for OpenTelemetry(ADOT) SDK 또는 X-Ray SDK 중 하나를 사용할 수 있습니다.

SDK를 사용하는 경우 애플리케이션을 계측할 때와 수집기 또는 에이전트를 구성할 때 모두 워크플로에 사용자 지정 사항을 추가할 수 있습니다. SDK를 사용하면 AWS Management Console에서는 수행할 수 없는 다음 작업을 수행할 수 있습니다.

- 사용자 지정 지표 게시 - 최대 1초 단위의 고해상도로 지표를 샘플링하고, 여러 차원을 사용하여 지표에 대한 정보를 추가하며, 데이터 포인트를 통계 집합에 집계할 수 있습니다.
- 수집기 사용자 지정 - 수신기, 프로세서, 내보내기 도구, 커넥터 등 수집기의 모든 부분에 대한 구성을 사용자 지정할 수 있습니다.
- 계측 사용자 지정 - 세그먼트 및 하위 세그먼트를 사용자 지정하고, 사용자 지정 키-값 페어를 속성으로 추가하며, 사용자 지정 지표를 생성할 수 있습니다.
- 샘플링 규칙을 프로그래밍 방식으로 생성하고 업데이트할 수 있습니다.

AWS 보안 및 최적화 계층이 추가된 표준화된 SDKADOT를 유연하게 사용하려면 OpenTelemetry SDK를 사용합니다. AWS Distro for OpenTelemetry(ADOT) SDK는 코드를 다시 계측할 필요 없이 다른 공급업체 및 비AWS 서비스의 백엔드와 통합할 수 있는 공급업체에 구애받지 않는 패키지입니다.

이미 X-Ray SDK를 사용하고 있으며, AWS 백엔드와만 통합하고 X-Ray 또는 애플리케이션 코드와의 상호 작용 방식은 변경하지 않으려는 경우 X-Ray SDK를 사용합니다.

각 기능에 대한 자세한 내용은 AWS Distro for OpenTelemetry와 X-Ray SDKs 중에서 선택 단원을 참조하세요.

ADOT SDK 사용

ADOT SDK는 백엔드 서비스로 데이터를 전송하는 오픈 소스 APIs, 라이브러리 및 에이전트 세트입니다. ADOT는에서 지원되며 여러 백엔드 및 에이전트와 AWS통합되고 OpenTelemetry 커뮤니티에서 유지 관리하는 많은 오픈 소스 라이브러리를 제공합니다. 애플리케이션을 계측하고 로그, 메타데이터, 지표 및 트레이스를 수집하려면 ADOT SDK를 사용하세요. ADOT를 사용하여 서비스를 모니터링하고 CloudWatch의 지표를 기반으로 경보를 설정할 수 있습니다.

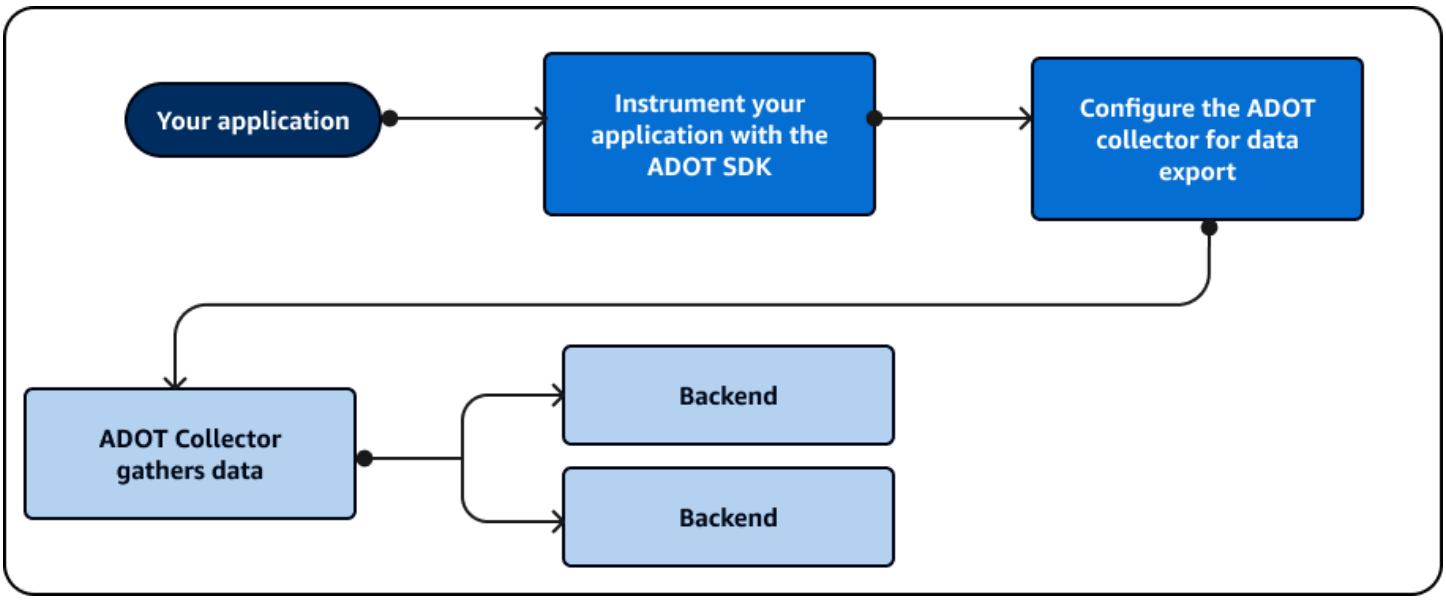
ADOT SDK를 사용하는 경우 에이전트와 함께 다음 옵션을 사용할 수 있습니다.

- CloudWatch 에이전트와 함께 ADOT SDK 사용 - 권장
- ADOT 수집기와 함께 ADOT SDK 사용 - 보안 및 최적화 AWS 계층과 함께 공급업체에 구애받지 않는 소프트웨어를 사용하려는 경우 권장됩니다.

ADOT SDK를 사용하려면 다음을 수행합니다.

- ADOT SDK를 사용하여 애플리케이션을 계측합니다. 자세한 내용은 ADOT 기술 설명서에서 사용 중인 프로그래밍 언어에 대한 설명서를 참조하세요.
- 수집한 데이터를 전송할 위치를 알려주도록 ADOT Collector를 구성합니다.

ADOT 수집기가 데이터를 수신한 후 ADOT 구성에서 지정한 백엔드로 데이터를 전송합니다.는 다음 다이어그램과 AWS같이 외부 공급업체를 포함하여 여러 백엔드로 데이터를 전송할 ADOT 수 있습니다.



AWS 는 정기적으로를 업데이트ADOT하여 기능을 추가하고 OpenTelemetry 프레임워크에 맞게 조정합니다. ADOT 개발을 위한 업데이트와 향후 계획은 로드맵의 일부로서 공개되어 있습니다. ADOT는 다음과 같은 여러 프로그래밍 언어를 지원합니다.

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

Python을 사용하는 경우 ADOT에서 애플리케이션을 자동으로 계측할 수 있습니다. 사용을 시작하려면 소개 및 AWS Distro for OpenTelemetry Collector 시작하기를 ADOT참조하세요.

X-Ray SDK 사용

X-Ray SDK는 백엔드 서비스로 데이터를 AWS 전송하는 AWS APIs 및 라이브러리 세트입니다. X-Ray SDK를 사용하면 애플리케이션을 계측하고 트레이스 데이터를 수집할 수 있습니다. X-Ray SDK를 사용하여 로그 또는 지표 데이터를 수집할 수는 없습니다.

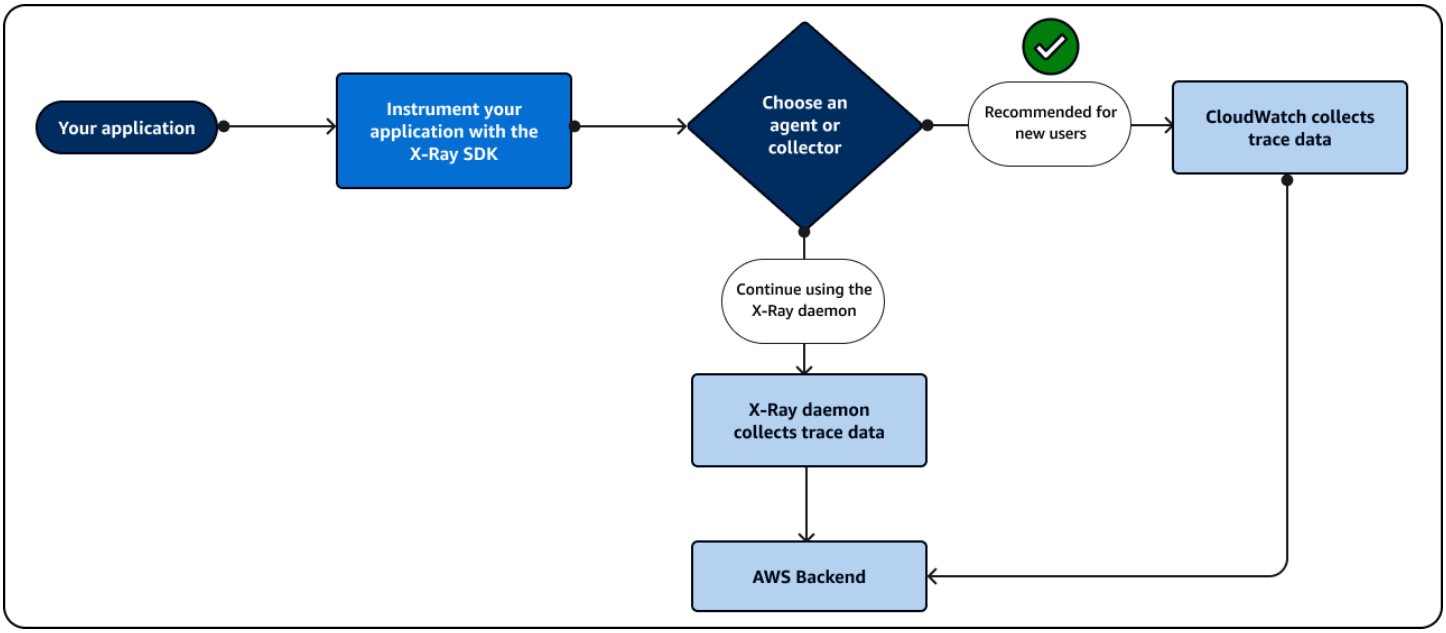
X-Ray SDK를 사용하는 경우 에이전트와 함께 다음 옵션을 사용할 수 있습니다.

- AWS X-Ray 데몬과 함께 X-Ray SDK 사용 - 애플리케이션 코드를 업데이트하지 않으려는 경우 이 옵션을 사용합니다.
- CloudWatch 에이전트와 함께 X-Ray SDK 사용 - (권장) CloudWatch 에이전트는 X-Ray SDK와 호환됩니다.

X-Ray SDK를 사용하려면 다음을 수행합니다.

- X-Ray SDK를 사용하여 애플리케이션을 계측합니다.
- 수집한 데이터를 전송할 위치를 알려주도록 수집기를 구성합니다. 트레이스 정보는 CloudWatch 에이전트 또는 X-Ray 대몬을 사용하여 수집할 수 있습니다.

수집기 또는 에이전트가 데이터를 수신하면 에이전트 구성에서 지정한 AWS 백엔드로 데이터를 전송합니다. X-Ray SDK는 다음 다이어그램과 같이 AWS 백엔드로만 데이터를 전송할 수 있습니다.



Java를 사용하는 경우 X-Ray SDK를 사용하여 애플리케이션을 자동으로 계측할 수 있습니다. X-Ray SDK 사용을 시작하려면 다음 프로그래밍 언어와 관련된 라이브러리를 참조하세요.

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

콘솔 사용

최소한의 코딩이 필요한 그래픽 사용자 인터페이스(GUI)를 원하는 경우 콘솔을 사용합니다. X-Ray를 처음 사용하는 사용자는 미리 빌드된 시각화를 사용하고 기본 작업을 수행하여 빠르게 시작할 수 있습니다. 콘솔에서 직접 다음을 수행할 수 있습니다.

- X-Ray 활성화하기
- 애플리케이션 성능에 대한 간략한 요약 확인
- 애플리케이션 상태 확인
- 상위 수준 오류 식별
- 기본 트레이스 요약 확인

Amazon CloudWatch 콘솔(https://console.aws.amazon.com/cloudwatch/) 또는 X-Ray 콘솔(https://console.aws.amazon.com/xray/home)을 사용하여 X-Ray와 상호 작용할 수 있습니다.

Amazon CloudWatch 콘솔 사용

CloudWatch 콘솔에는 더 쉽게 사용할 수 있도록 X-Ray 콘솔에서 다시 설계된 새로운 X-Ray 기능이 포함되어 있습니다. CloudWatch 콘솔을 사용하는 경우 X-Ray 트레이스 데이터와 함께 CloudWatch 로그와 지표를 확인할 수 있습니다. CloudWatch 콘솔을 사용하여 다음을 포함한 데이터를 확인하고 분석할 수 있습니다.

- X-Ray 트레이스 - 요청을 처리할 때 애플리케이션과 연관된 트레이스를 확인, 분석 및 필터링할 수 있습니다. 이러한 트레이스를 사용하여 지연 시간이 높은 항목을 찾고 오류를 디버깅하며 애플리케이션 워크플로를 최적화할 수 있습니다. 트레이스 맵과 서비스 맵을 확인하여 애플리케이션 워크플로를 시각적으로 확인할 수 있습니다.

- 로그 - 애플리케이션에서 생성하는 로그를 확인, 분석 및 필터링할 수 있습니다. 로그를 사용하여 오류를 해결하고 특정 로그 값을 기반으로 모니터링을 설정할 수 있습니다.
- 지표 - 리소스가 제공하는 지표를 사용하여 애플리케이션 성능을 측정 및 모니터링하거나 직접 지표를 생성할 수 있습니다. 이러한 지표는 그래프와 차트로 확인할 수 있습니다.
- 네트워크 및 인프라 모니터링 - 주요 네트워크의 중단과 컨테이너화된 애플리케이션, 기타 AWS 서비스 및 클라이언트를 포함한 인프라의 상태 및 성능을 모니터링합니다.
- 다음 X-Ray 콘솔 사용하기 섹션에 나열된 X-Ray 콘솔의 모든 기능

CloudWatch 콘솔에 대한 자세한 내용은 Amazon CloudWatch 시작하기를 참조하세요.

Amazon CloudWatch 콘솔(https://console.aws.amazon.com/cloudwatch/)에 로그인합니다.

X-Ray 콘솔 사용하기

X-Ray 콘솔은 애플리케이션 요청에 대한 분산 추적 기능을 제공합니다. 더 간단한 콘솔 환경을 원하거나 애플리케이션 코드를 업데이트하지 않으려면 X-Ray 콘솔을 사용합니다. AWS 는 더 이상 X-Ray 콘솔을 개발하지 않습니다. X-Ray 콘솔에는 계측 애플리케이션을 위한 다음과 같은 기능이 포함되어 있습니다.

- 인사이트 - 애플리케이션 성능의 이상 징후를 자동으로 감지하고 근본적인 원인을 찾아냅니다. 인사이트는 CloudWatch 콘솔의 인사이트 아래에 포함되어 있습니다. 자세한 내용은 X-Ray 콘솔 사용하기에서 X-Ray 인사이트 사용을 참조하세요.
- 서비스 맵 - 애플리케이션의 그래픽 구조와 클라이언트, 리소스, 서비스 및 종속성과의 연결을 볼 수 있습니다.
- 트레이스 - 애플리케이션이 요청을 처리할 때 생성되는 트레이스의 개요를 확인할 수 있습니다. 트레이스 데이터를 사용하여 HTTP 응답 및 응답 시간을 비롯한 기본 지표에 대한 애플리케이션의 성능을 파악할 수 있습니다.
- 분석 - 응답 시간 분포 그래프를 사용하여 트레이스 데이터를 해석, 탐색 및 분석할 수 있습니다.
- 구성 - 사용자 지정 트레이스를 생성하여 다음에 대한 기본 구성을 변경할 수 있습니다.
  - 샘플링 - 애플리케이션에서 트레이스 정보를 샘플링하는 빈도를 정의하는 규칙을 생성합니다. 자세한 내용은 X-Ray 콘솔 사용하기에서 샘플링 규칙 구성을 참조하세요.
  - 암호화 - AWS Key Management Service를 통해 감사하거나 비활성화할 수 있는 키를 사용하여 저장 데이터를 암호화합니다.
  - 그룹 - 필터 표현식을 사용하여 URL 이름이나 응답 시간과 같은 공통된 특성을 가진 트레이스 그룹을 정의할 수 있습니다. 자세한 내용은 그룹 구성을 참조하세요.

X-Ray 콘솔(https://console.aws.amazon.com/xray/home)에 로그인합니다.

X-Ray 콘솔 살펴보기

X-Ray 콘솔을 사용하여 애플리케이션이 제공하는 요청에 대한 서비스 맵 및 관련 트레이스를 확인하고, 트레이스가 X-Ray로 전송되는 방식에 영향을 주는 그룹 및 샘플링 규칙을 구성할 수 있습니다.

> **Note**
>
> X-Ray 서비스 맵과 CloudWatch ServiceLens 맵은 Amazon CloudWatch 콘솔 내에서 X-Ray 트레이스 맵으로 통합되었습니다. CloudWatch 콘솔을 열고 왼쪽 탐색 창의 X-Ray 트레이스에서 트레이스 맵을 선택합니다.
>
> 이제 CloudWatch에 애플리케이션 서비스, 클라이언트, Synthetics Canary 및 서비스 종속성을 검색하고 모니터링할 수 있는 Application Signals가 포함됩니다. Application Signals를 사용하여 서비스의 목록 또는 시각적 맵을 확인하고, 서비스 수준 목표(SLO)를 기준으로 상태 지표를 확인하고, 더 자세한 문제 해결을 위해 상관관계가 있는 X-Ray 트레이스를 드릴다운할 수 있습니다.

기본 X-Ray 콘솔 페이지는 애플리케이션에서 생성한 트레이스 데이터를 바탕으로 X-Ray가 생성하는 JSON 서비스 그래프를 시각적으로 표현한 트레이스 맵입니다. 맵은 요청을 처리하는 계정 내 각 애플리케이션에 대한 서비스 노드, 요청의 오리진을 나타내는 업스트림 클라이언트 노드, 애플리케이션이 요청을 처리하는 동안 사용하는 웹 서비스 및 리소스를 나타내는 다운스트림 서비스 노드로 구성됩니다. 추가 페이지에서 추적 및 추적 세부 정보를 보고 그룹 및 샘플링 규칙을 구성할 수 있습니다.

다음 단원에서 X-Ray의 콘솔 환경을 살펴보고 CloudWatch 콘솔과 비교해 보세요.

X-Ray 및 CloudWatch 콘솔 살펴보기

- X-Ray 트레이스 맵 사용
- 트레이스 및 트레이스 세부 정보 보기
- 필터 표현식 사용
- 교차 계정 추적
- 이벤트 기반 애플리케이션 추적
- 지연 시간 히스토그램 사용
- X-Ray 인사이트 사용
- 분석 콘솔과 상호 작용

- 그룹 구성
- 샘플링 규칙 구성
- 콘솔 심층 연결

X-Ray 트레이스 맵 사용

X-Ray 트레이스 맵을 보고 오류가 발생한 서비스, 지연 시간이 긴 연결 또는 실패한 요청에 대한 트레이스를 식별할 수 있습니다.

> **Note**
>
> 이제 CloudWatch에 애플리케이션 서비스, 클라이언트, Synthetics Canary 및 서비스 종속성을 검색하고 모니터링할 수 있는 Application Signals가 포함됩니다. Application Signals를 사용하여 서비스의 목록 또는 시각적 맵을 확인하고, 서비스 수준 목표(SLO)를 기준으로 상태 지표를 확인하고, 더 자세한 문제 해결을 위해 상관관계가 있는 X-Ray 트레이스를 드릴다운할 수 있습니다.
>
> X-Ray 서비스 맵과 CloudWatch ServiceLens 맵은 Amazon CloudWatch 콘솔 내의 X-Ray 트레이스 맵에 결합됩니다. CloudWatch 콘솔을 열고 왼쪽 탐색 창의 X-Ray 트레이스에서 트레이스 맵을 선택합니다.

추적 맵 보기

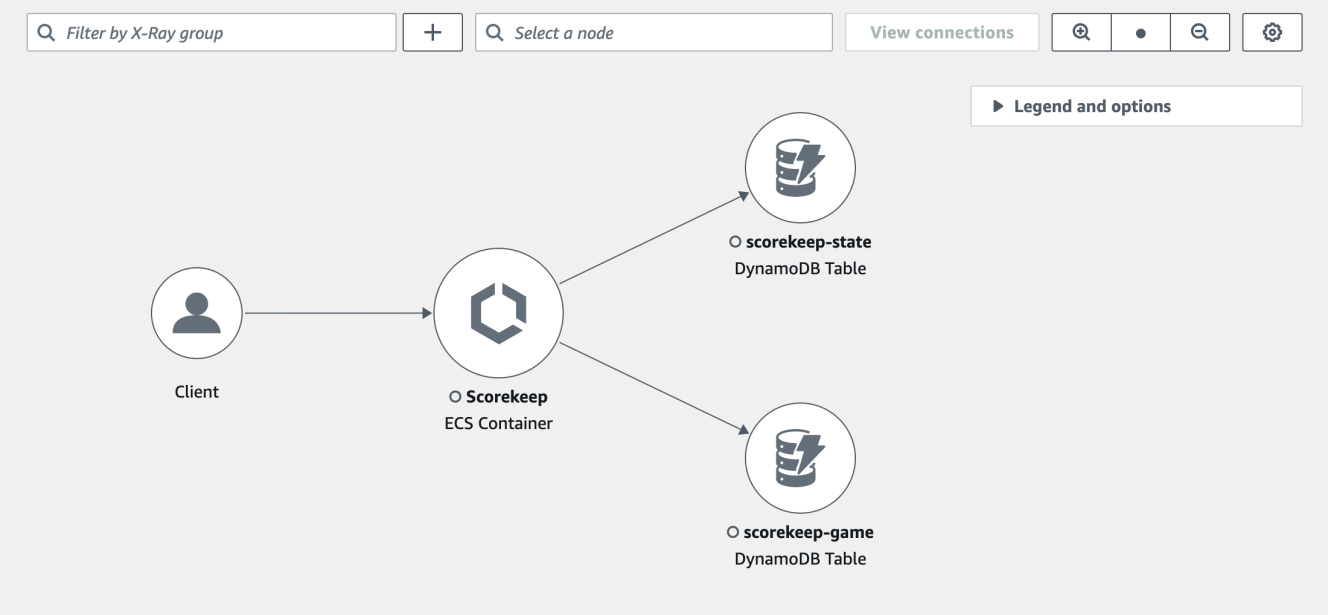
트레이스 맵은 애플리케이션에서 생성되는 트레이스 데이터를 시각적으로 표현한 것입니다. 맵에는 요청을 처리하는 서비스 노드, 요청의 출처를 나타내는 업스트림 클라이언트 노드, 요청을 처리하는 동안 애플리케이션에서 사용하는 웹 서비스 및 리소스를 나타내는 다운스트림 서비스 노드가 표시됩니다.

트레이스 맵은 Amazon SQS 및 Lambda를 사용하는 이벤트 기반 애플리케이션 전반의 트레이스에 대한 연관성을 표시합니다. 자세한 내용은 이벤트 중심 애플리케이션 트레이싱을 참조하세요. 또한 트레이스 맵은 교차 계정 추적을 지원하여 여러 계정의 노드를 단일 맵에 표시합니다.

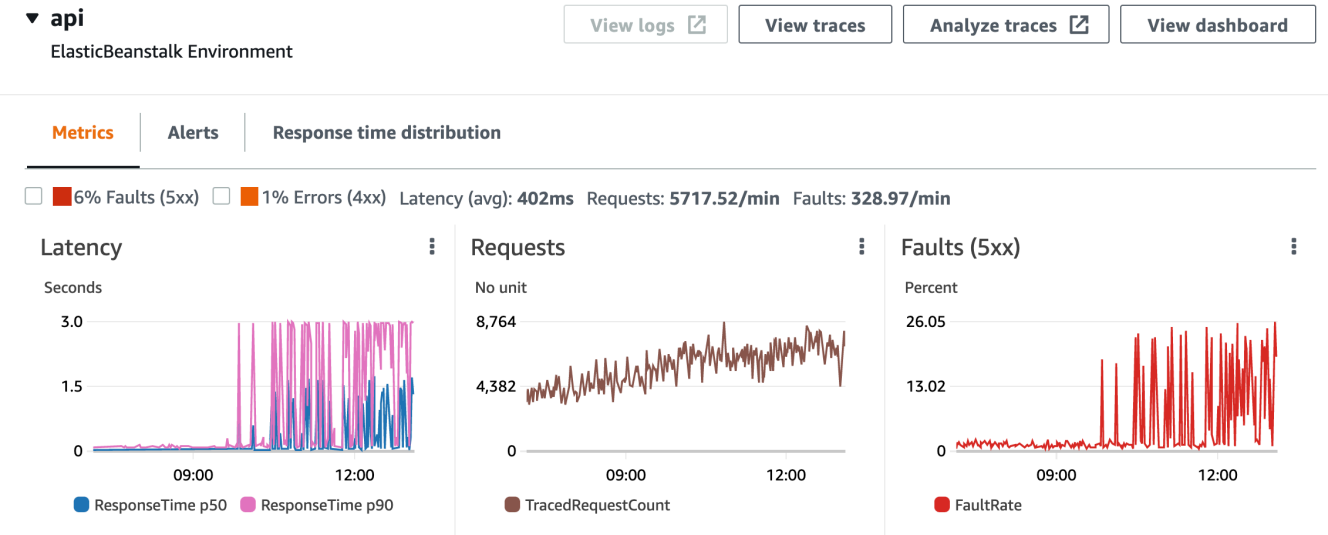
CloudWatch console

CloudWatch 콘솔에서 트레이스 맵을 보려면

1. CloudWatch 콘솔을 엽니다. 왼쪽 탐색 창의 X-Ray 트레이스 섹션에서 트레이스 맵을 선택합니다.



2. 노드에 대한 요청을 보려는 서비스 노드 또는 그 사이를 이동하는 요청을 보려는 두 노드 간 에지를 선택합니다.
3. 트레이스 맵 아래에 지표, 알림 및 응답 시간 분포에 대한 탭을 비롯한 추가 정보가 표시됩니다. 지표 탭에서 각 그래프 내 범위를 선택하여 드릴다운하여 자세한 내용을 보거나, 결함 또는 오류 옵션을 선택하여 트레이스를 필터링합니다. 응답 시간 분포 탭에서 그래프 내의 범위를 선택하여 응답 시간별로 추적을 필터링합니다.



4. 트레이스 보기를 선택하여 트레이스를 보거나, 필터가 적용된 경우 필터링된 트레이스 보기를 선택합니다.
5. 선택한 노드와 관련된 CloudWatch 로그를 보려면 로그 보기 를 선택합니다. 모든 트레이스 맵 노드가 로그 보기를 지원하는 것은 아닙니다. 자세한 내용은 CloudWatch 로그 문제 해결을 참조하세요.

트레이스 맵은 각 노드 내의 문제를 색상별로 표시합니다.

- 빨간색은 서버 장애를 의미합니다(500 시리즈 오류)
- 노란색은 클라이언트 오류를 의미합니다(400 시리즈 오류)
- 보라색은 병목 오류를 의미합니다(429 요청 과다)

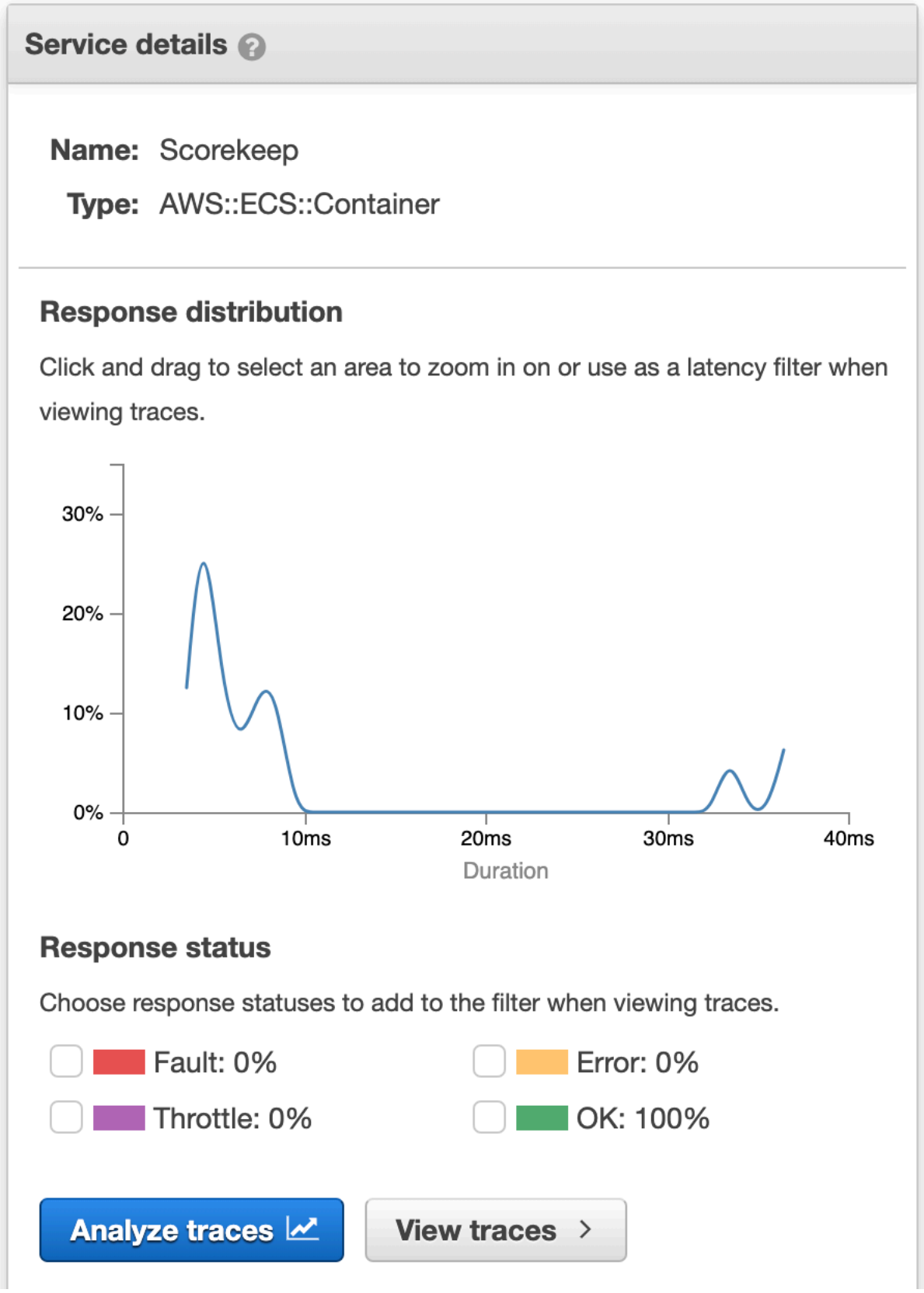
트레이스 맵이 큰 경우 화면의 컨트롤이나 마우스를 사용하여 맵을 확대/축소하거나 이동할 수 있습니다.

X-Ray console

서비스 맵을 보려면

1. X-Ray 콘솔을 엽니다. 기본적으로 서비스 맵이 표시됩니다. 왼쪽 탐색 창에서 서비스 맵을 선택할 수도 있습니다.

2. 노드에 대한 요청을 보려는 서비스 노드 또는 그 사이를 이동하는 요청을 보려는 두 노드 간 에지를 선택합니다.
3. 응답 분포 히스토그램을 사용하여 트레이스를 기간별로 필터링하고 트레이스를 보려는 상태 코드를 선택합니다. 그런 다음 [View traces]를 선택하여 필터 표현식이 적용된 트레이스 목록을 엽니다.


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

이 서비스는 오류 및 장애 호출 성공 비율을 토대로 각 노드의 색상을 다르게 표시해 노드의 상태를 보여줍니다:

- 녹색은 성공적인 호출을 의미합니다
- 빨간색은 서버 장애를 의미합니다(500 시리즈 오류)
- 노란색은 클라이언트 오류를 의미합니다(400 시리즈 오류)
- 보라색은 병목 오류를 의미합니다(429 요청 과다)

서비스 맵이 큰 경우 화면의 컨트롤이나 마우스를 사용하여 맵을 확대/축소하거나 이동할 수 있습니다.

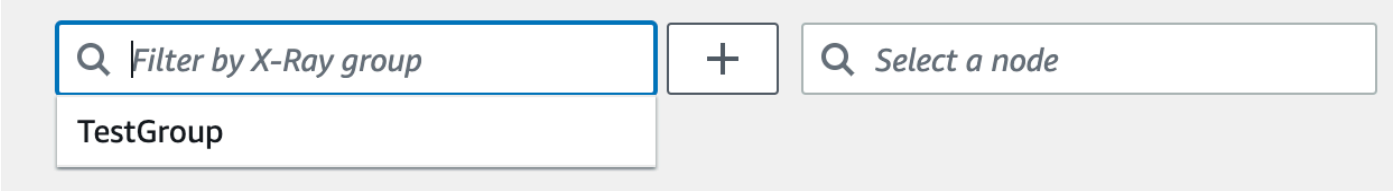
> **Note**
>
> X-Ray 트레이스 맵에는 최대 10,000개의 노드가 표시될 수 있습니다. 드물지만 총 서비스 노드 수가 이 한도를 초과할 경우 오류가 발생하여 콘솔에 전체 트레이스 맵이 표시되지 않을 수 있습니다.

그룹으로 트레이스 맵 필터링

필터 표현식을 사용하여 그룹에 트레이스를 허용하는 기준을 정의할 수 있습니다. 다음 단계에 따라 트레이스 맵에 해당 특정 그룹을 표시합니다.

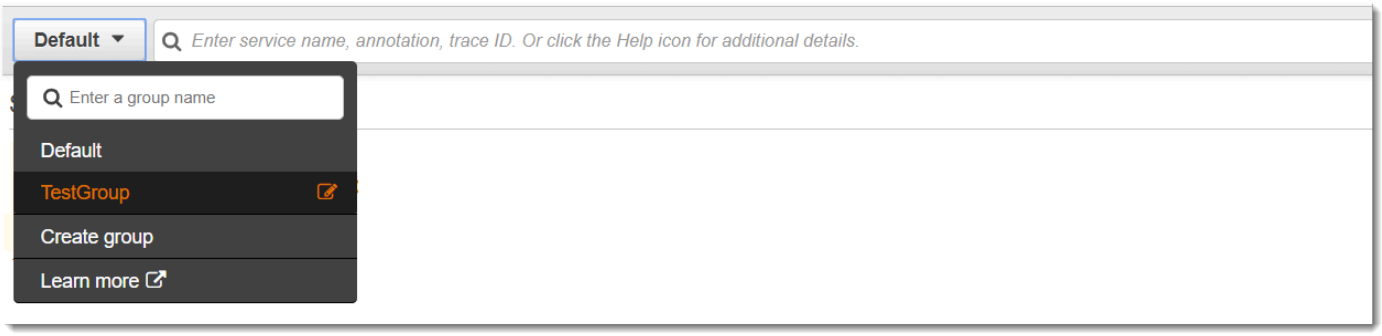
CloudWatch console

트레이스 맵의 왼쪽 상단에 있는 그룹 필터에서 그룹 이름을 선택합니다.



X-Ray console

검색 창 왼쪽에 있는 드롭다운 메뉴에서 그룹 이름을 선택합니다.



이제 서비스 맵이 필터링되어 선택한 그룹의 필터 표현식과 일치하는 추적을 표시합니다.

트레이스 맵 범례 및 옵션

트레이스 맵에는 범례와 맵 표시를 사용자 지정할 수 있는 몇 가지 옵션이 포함되어 있습니다.

CloudWatch console

지도 오른쪽 상단의 범례 및 옵션 드롭다운을 선택합니다. 다음을 포함하여 노드 내에 표시할 항목을 선택합니다:

- 지표는 선택한 시간 범위 동안 분당 평균 응답 시간과 전송된 추적 횟수를 표시합니다.
- 노드는 각 노드 내의 서비스 아이콘이 표시됩니다.

맵 오른쪽 상단의 톱니바퀴 아이콘을 통해 액세스할 수 있는 기본 설정 창에서 추가 맵 설정을 선택합니다. 이러한 설정에는 각 노드의 크기를 결정하는 데 사용되는 메트릭과 맵에 표시할 카나리아를 선택하는 것이 포함됩니다.

X-Ray console

맵 오른쪽 상단의 맵 레전드 링크를 선택하여 서비스 맵 범례를 표시합니다. 서비스 맵 오른쪽 하단에서 다음을 포함한 트레이스 맵 옵션을 선택할 수 있습니다.

- 서비스 아이콘은 각 노드 내에 표시되는 내용을 전환하여 서비스 아이콘 또는 선택한 시간 범위 동안 분당 평균 응답 시간 및 전송된 추적 횟수를 표시합니다.
- 노드 크기 조정: 없음은 모든 노드의 크기를 동일하게 설정합니다.
- 노드 크기 조정: Health는 오류, 결함 또는 병목 현상이 발생한 요청을 포함하여 영향을 받는 요청 수를 기준으로 노드 크기를 조정합니다.
- 노드 크기 조정: 트래픽 총 요청 수를 기준으로 노드의 크기를 조정합니다.

트레이스 및 트레이스 세부 정보 보기

X-Ray 콘솔의 트레이스 페이지를 사용하여 URL, 응답 코드 또는 트레이스 요약의 다른 데이터를 기준으로 트레이스를 검색할 수 있습니다. 트레이스 목록에서 트레이스를 선택하면 트레이스 세부 정보 페이지에 선택한 트레이스와 연관된 서비스 노드 맵과 트레이스 세그먼트의 타임라인이 표시됩니다.

추적 보기

CloudWatch console

CloudWatch 콘솔에서 트레이스를 보려면

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/cloudwatch/:// https://https://://https://://https://://CloudWatch://://https://://https://://https://https://://https://:// https
2. 왼쪽 탐색 창에서 X-Ray 트레이스를 선택한 다음 트레이스를 선택합니다. 그룹으로 필터링하거나 필터 표현식을 입력할 수 있습니다. 이 경우 페이지 하단의 트레이스 섹션에 표시되는 트레이스가 필터링됩니다.

   또는 서비스 맵을 사용하여 특정 서비스 노드로 이동한 다음 트레이스를 볼 수 있습니다. 이 경우 쿼리가 이미 적용된 트레이스 페이지가 열립니다.
3. 쿼리 세분화 섹션에서 쿼리를 구체화하세요. 공통 속성을 기준으로 트레이스를 필터링하려면 쿼리 세분화 기준 옆의 아래쪽 화살표에서 옵션을 선택합니다. 옵션에는 다음 사항이 포함됩니다.

   - 노드 - 서비스 노드로 트레이스를 필터링합니다.
   - 리소스 ARN - 트레이스와 연결된 리소스로 트레이스를 필터링합니다. 이러한 리소스의 예로는 Amazon Elastic Compute Cloud(Amazon EC2) 인스턴스, AWS Lambda 함수 또는 Amazon DynamoDB 테이블이 있습니다.
   - 사용자 - 사용자 ID로 트레이스를 필터링합니다.
   - 오류 근본 원인 메시지 - 오류 근본 원인으로 트레이스를 필터링합니다.
   - URL - 애플리케이션에서 사용하는 URL 경로로 트레이스를 필터링합니다.
   - HTTP 상태 코드 - 애플리케이션에서 반환하는 HTTP 상태 코드로 트레이스를 필터링합니다. 사용자 지정 응답 코드를 지정하거나 다음 중에서 선택할 수 있습니다.
     - `200` - 요청이 성공했습니다.
     - `401` - 요청에 유효한 인증 자격 증명이 부족합니다.
     - `403` – 요청에 유효한 권한이 부족합니다.

- `404` - 서버에서 요청된 리소스를 찾을 수 없습니다.
- `500` - 서버에서 예기치 않은 조건이 발생하여 내부 오류가 발생했습니다.

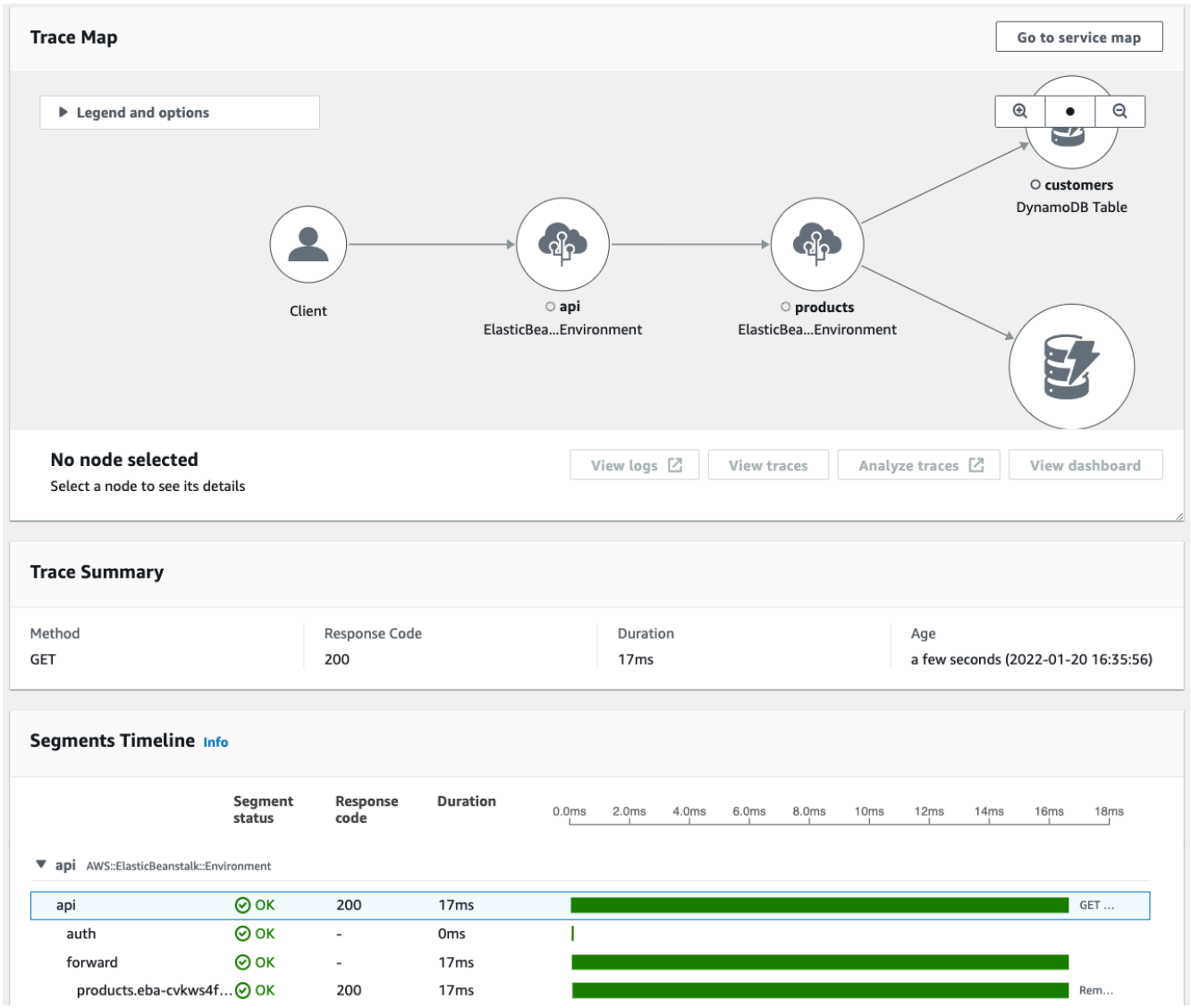
항목을 하나 이상 선택한 다음 쿼리에 추가를 선택하여 페이지 상단의 필터 표현식에 추가합니다.

4. 단일 트레이스를 찾으려면 쿼리 필드에 트레이스 ID를 직접 입력합니다. X-Ray 형식 또는 W3C(World Wide Web Consortium) 형식을 사용할 수 있습니다. 예를 들어 AWS Distro for OpenTelemetry를 사용하여 생성된 트레이스는 W3C 형식입니다.

   > **Note**
   >
   > W3C 형식의 트레이스 ID로 생성된 트레이스를 쿼리하면 일치하는 트레이스가 콘솔에 X-Ray 형식으로 표시됩니다. 예를 들어 `4efaaf4d1e8720b39541901950019ee5`를 W3C 형식으로 쿼리하면 동등한 X-Ray 형식인 `1-4efaaf4d-1e8720b39541901950019ee5`가 콘솔에 표시됩니다.

5. 페이지 하단의 트레이스 섹션에 일치하는 트레이스 목록을 표시하려면 언제든지 쿼리 실행을 선택하십시오.
6. 단일 트레이스에 대한 트레이스 세부 정보 페이지를 표시하려면 목록에서 트레이스 ID를 선택합니다.

   다음 이미지는 트레이스와 관련된 서비스 노드와 트레이스를 구성하는 세그먼트가 이동한 경로를 나타내는 노드 간의 엣지가 포함된 트레이스 맵을 보여줍니다. 트레이스 맵 다음에 트레이스 요약이 표시됩니다. 요약에는 샘플 `GET` 작업, 응답 코드, 트레이스가 실행되는 데 걸린 기간 및 요청 기간에 대한 정보가 포함되어 있습니다. 트레이스 세그먼트 및 하위 세그먼트의 기간을 보여주는 세그먼트 타임라인은 트레이스 요약 뒤에 표시됩니다.



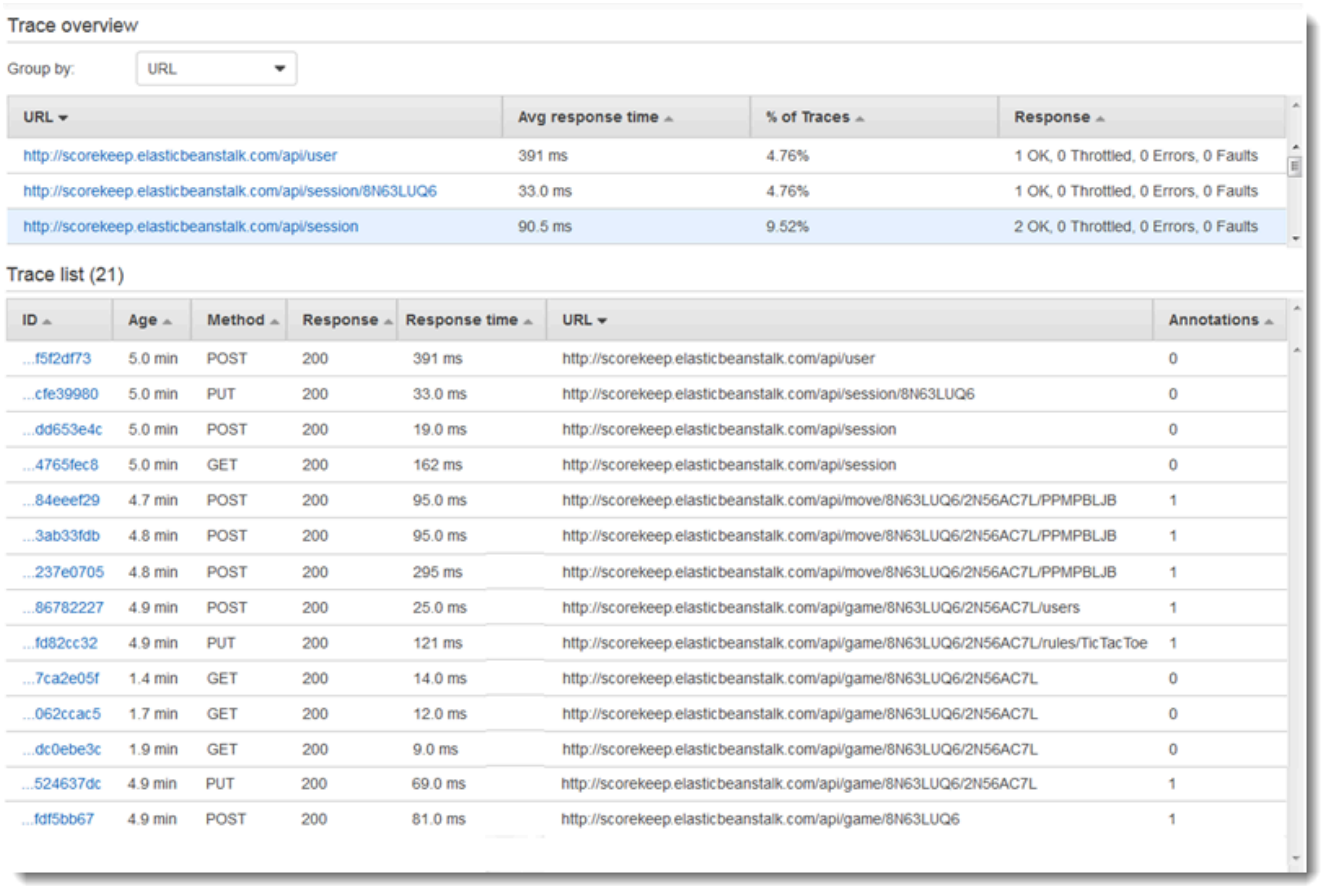
Amazon SQS 및 Lambda를 사용하는 이벤트 기반 애플리케이션이 있는 경우, 트레이스 맵에서 각 요청에 대한 트레이스를 연결된 형태로 볼 수 있습니다. 맵에서 메시지 생산자의 트레이스는 AWS Lambda 소비자의 트레이스와 연결되며 파선 엣지로 표시됩니다. 이벤트 기반 애플리케이션에 대한 자세한 내용은 이벤트 기반 애플리케이션 추적 단원을 참조하세요.

트레이스 및 트레이스 세부 정보 페이지에서는 교차 계정 추적을 지원합니다. 이를 통해 트레이스 목록과 단일 트레이스 맵 내에서 여러 계정의 트레이스를 나열할 수 있습니다.

X-Ray console

X-Ray 콘솔에서 트레이스를 보려면

1. X-Ray 콘솔의 트레이스 페이지를 엽니다. 트레이스 개요 패널에는 오류 근본 원인, ResourceARN, InstanceId를 포함한 일반적인 기능별로 그룹화된 트레이스 목록이 표시됩니다.
2. 공통 기능을 선택해 그룹화된 트레이스를 보려면 그룹화 기준 옆의 아래쪽 화살표를 확장합니다. 다음 그림은 AWS X-Ray 샘플 애플리케이션의 URL별로 그룹화된 트레이스의 트레이스 개요와 관련 트레이스 목록을 보여줍니다.



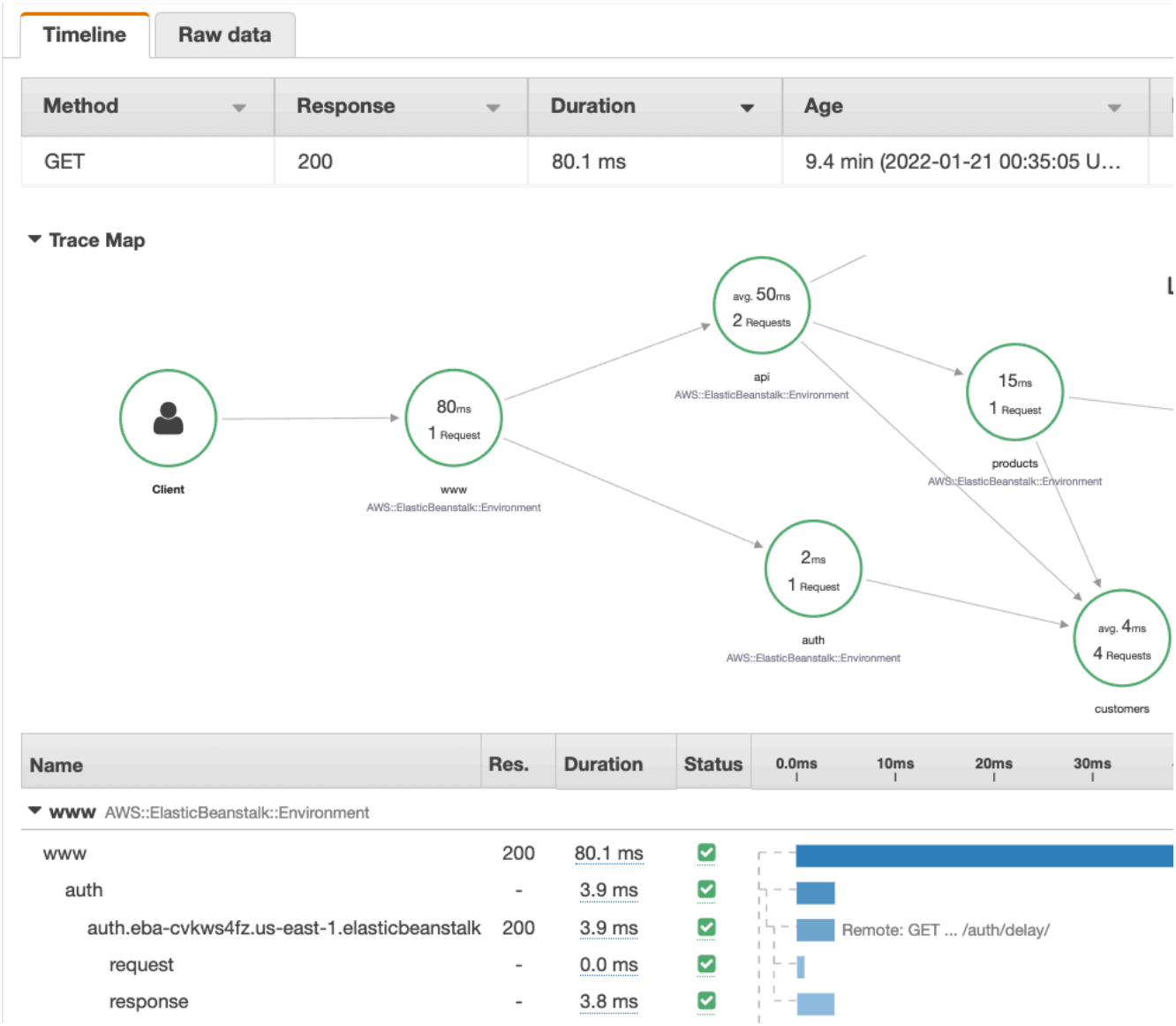
3. 트레이스 목록에서 확인할 트레이스의 ID를 선택합니다. 탐색 창에서 서비스 맵을 선택하여 특정 서비스 노드에 대한 트레이스를 볼 수도 있습니다. 그런 다음 해당 노드와 연관된 트레이스를 볼 수 있습니다.

   타임라인 탭에는 트레이스에 대한 요청 흐름이 표시되며 다음 정보가 포함됩니다.

   - 트레이스의 각 세그먼트에 대한 경로 맵
   - 세그먼트가 트레이스 맵의 노드에 도달하는 데 걸린 시간

- 트레이스 맵의 노드에 대한 요청 횟수

다음 그림은 샘플 애플리케이션에 대한 GET 요청과 관련된 트레이스 맵의 예시를 보여줍니다. 화살표는 각 세그먼트가 요청을 완료하기 위해 이동한 경로를 표시합니다. 서비스 노드에는 GET 요청 중에 이루어진 요청 횟수가 표시됩니다.



타임라인 탭에 대한 자세한 내용은 아래의 Exploring the trace timeline 단원을 참조하세요.

원시 데이터 탭에는 트레이스와 이를 구성하는 세그먼트 및 하위 세그먼트에 대한 정보가 JSON 형식으로 표시됩니다. 여기에는 다음 정보가 포함될 수 있습니다.

- 타임스탬프

- 고유 ID
- 세그먼트 또는 하위 세그먼트와 연관된 리소스
- 세그먼트 또는 하위 세그먼트의 소스 또는 오리진
- 애플리케이션 요청에 대한 추가 정보(예: HTTP 요청의 응답)

트레이스 타임라인 살펴보기

타임라인 섹션에는 세그먼트와 하위 세그먼트가 계층 구조로 표시되며, 그 옆에는 작업 완료 시간에 해당하는 가로 막대가 나타납니다. 목록의 첫 번째 항목이 세그먼트인데, 단일 요청의 서비스에 의해 기록된 모든 데이터를 나타냅니다. 하위 세그먼트는 들여쓰기되어 세그먼트 아래에 나열됩니다. 열에는 각 세그먼트에 대한 정보가 포함되어 있습니다.

CloudWatch console

CloudWatch 콘솔에서 세그먼트 타임라인은 다음 정보를 제공합니다.

- 첫 번째 열: 선택한 트레이스의 세그먼트 및 하위 세그먼트가 나열됩니다.
- 세그먼트 상태 열: 각 세그먼트 및 하위 세그먼트의 상태 결과가 나열됩니다.
- 응답 코드 열: 세그먼트 또는 하위 세그먼트의 브라우저 요청에 대한 HTTP 응답 상태 코드가 나열됩니다(제공되는 경우).
- 기간 열: 세그먼트 또는 하위 세그먼트가 실행된 시간이 나열됩니다.
- 호스팅 위치 열: 세그먼트 또는 하위 세그먼트가 실행되는 네임스페이스 또는 환경이 나열됩니다(해당되는 경우). 자세한 내용은 수집된 측정기준 및 측정기준 조합을 참조하세요.
- 마지막 열: 타임라인의 다른 세그먼트 또는 하위 세그먼트와 비교하여 세그먼트 또는 하위 세그먼트가 실행된 기간에 해당하는 가로 막대가 표시됩니다.

서비스 노드별로 세그먼트 및 하위 세그먼트 목록을 그룹화하려면 노드별 그룹화를 켭니다.

X-Ray console

트레이스 세부 정보 페이지에서 타임라인 탭을 선택하면 트레이스를 구성하는 각 세그먼트와 하위 세그먼트의 타임라인을 확인할 수 있습니다.

X-Ray 콘솔에서 타임라인은 다음 정보를 제공합니다.

- 이름 열: 트레이스의 세그먼트 및 하위 세그먼트의 이름이 나열됩니다.

- 응답 열: 세그먼트 또는 하위 세그먼트의 브라우저 요청에 대한 HTTP 응답 상태 코드가 나열됩니다(제공되는 경우).
- 기간 열: 세그먼트 또는 하위 세그먼트가 실행된 시간이 나열됩니다.
- 상태 열: 세그먼트 또는 하위 세그먼트 상태의 결과가 나열됩니다.
- 마지막 열: 타임라인의 다른 세그먼트 또는 하위 세그먼트와 비교하여 세그먼트 또는 하위 세그먼트가 실행된 기간에 해당하는 가로 막대가 표시됩니다.

콘솔에서 타임라인을 생성하는 데 사용하는 원시 트레이스 데이터를 보려면 원시 데이터 탭을 선택합니다. 원시 데이터에는 트레이스와 이를 구성하는 세그먼트 및 하위 세그먼트에 대한 정보가 JSON 형식으로 표시됩니다. 여기에는 다음 정보가 포함될 수 있습니다.

- 타임스탬프
- 고유 ID
- 세그먼트 또는 하위 세그먼트와 연관된 리소스
- 세그먼트 또는 하위 세그먼트의 소스 또는 오리진
- 애플리케이션 요청에 대한 추가 정보(예: HTTP 요청의 응답)

계측된 AWS SDK, HTTP또는 SQL 클라이언트를 사용하여 외부 리소스를 호출하면 X-Ray SDK는 하위 세그먼트를 자동으로 기록합니다. 또한 X-Ray SDK를 사용하여 함수 또는 코드 블록에 대해 사용자 지정 하위 세그먼트를 기록할 수도 있습니다. 사용자 지정 하위 세그먼트가 열려 있는 동안 기록된 추가 하위 세그먼트는 사용자 지정 하위 세그먼트의 하위 항목이 됩니다.

세그먼트 세부 정보 보기

트레이스 타임라인에서 세부 정보를 보려는 세그먼트 이름을 선택합니다.

세그먼트 세부 정보 패널에는 개요, 리소스, 주석, 메타데이터, 예외, SQL 탭이 표시됩니다. 다음 사항이 적용됩니다.

- 개요 탭에 요청 및 응답에 대한 정보가 표시됩니다. 이름, 시작 시간, 종료 시간, 기간, 요청 URL, 요청 작업, 요청 응답 코드, 오류 및 결함 등의 정보가 포함됩니다.
- 세그먼트의 리소스 탭에는 X-Ray SDK의 정보와 애플리케이션을 실행하는 AWS 리소스에 대한 정보가 표시됩니다. X-Ray SDK용 Amazon EC2 AWS Elastic Beanstalk또는 Amazon ECS 플러그인을 사용하여 서비스별 리소스 정보를 기록합니다. 플러그인에 대한 자세한 내용은 Java용 X-Ray SDK 구성에서 서비스 플러그인 단원을 참조하세요.

- 나머지 탭에는 세그먼트에 대해 기록된 주석, 메타데이터 및 예외가 표시됩니다. 계측된 요청에서 발생하는 예외는 자동으로 캡처됩니다. 주석 및 메타데이터에는 X-Ray SDK에서 제공하는 작업을 사용하여 기록되는 추가 정보가 포함됩니다. 세그먼트에 주석 또는 메타데이터를 추가하려면 X-Ray SDK를 사용합니다. 자세한 내용은에서 SDK를 사용하여 AWS X-Ray SDKs용 애플리케이션 계측 AWS X-Ray.

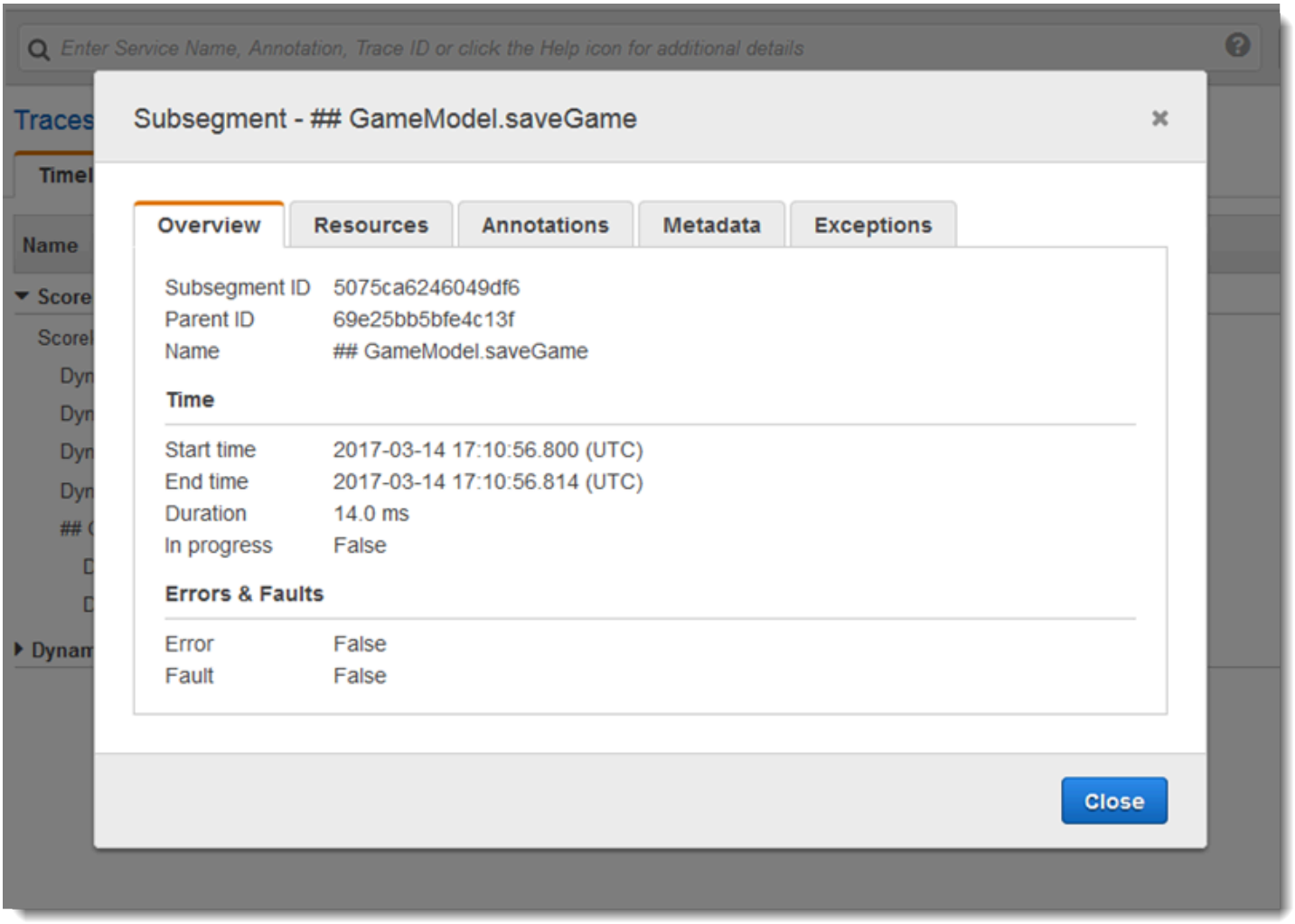
하위 세그먼트 세부 정보 보기

트레이스 타임라인에서 세부 정보를 보려는 하위 세그먼트 이름을 선택합니다.

- 개요 탭에는 요청 및 응답에 대한 정보가 포함되어 있습니다. 여기에는 이름, 시작 시간, 종료 시간, 기간, 요청 URL, 요청 작업, 요청 응답 코드, 오류 및 결함 등이 포함됩니다. 구성된 클라이언트에 의해 생성된 하위 세그먼트의 경우, 개요 탭에 애플리케이션 관점에서의 요청 및 응답에 대한 정보가 포함됩니다.
- 하위 세그먼트의 리소스 탭에는 하위 세그먼트를 실행하는 데 사용된 AWS 리소스에 대한 세부 정보가 표시됩니다. 예를 들어 리소스 탭에는 AWS Lambda 함수 ARN, DynamoDB 테이블에 대한 정보, 호출된 작업 및 요청 ID가 포함될 수 있습니다.
- 나머지 탭에는 하위 세그먼트에 대해 기록된 주석, 메타데이터 및 예외가 표시됩니다. 계측된 요청에서 발생하는 예외는 자동으로 캡처됩니다. 주석 및 메타데이터에는 X-Ray SDK에서 제공하는 작업을 사용하여 기록되는 추가 정보가 포함됩니다. 세그먼트에 주석 또는 메타데이터를 추가하려면 X-Ray SDK를 사용합니다. 자세한 내용은 용 애플리케이션 계측 AWS X-Ray에서 AWS X-Ray SDK로 애플리케이션 계측 아래에 나열된 언어별 링크를 참조하세요.

사용자 지정 하위 세그먼트의 경우, 개요 탭에 하위 세그먼트의 이름이 표시됩니다. 이 이름은 이 하위 세그먼트가 기록하는 코드 영역이나 함수를 지정하는 데 설정할 수 있습니다. 자세한 내용은에서 SDK를 사용하여 AWS X-Ray SDKsJava용 X-Ray SDK를 사용하여 사용자 지정 하위 세그먼트 생성하기.

다음 이미지는 사용자 지정 하위 세그먼트의 개요 탭을 보여줍니다. 개요에는 하위 세그먼트 ID, 상위 ID, 이름, 시작 및 종료 시간, 기간, 상태 및 오류 또는 결함이 포함되어 있습니다.



사용자 지정 하위 세그먼트의 메타데이터 탭에는 해당 하위 세그먼트에서 사용하는 리소스에 대한 정보가 JSON 형식으로 포함되어 있습니다.

필터 표현식 사용

필터 표현식을 사용하여 특정 요청, 서비스, 두 서비스 간의 연결(엣지) 또는 조건을 만족하는 요청에 대해 트레이스 맵 또는 트레이스를 표시할 수 있습니다. X-Ray는 요청 헤더, 응답 상태 및 원래 세그먼트의 인덱싱된 필드에 포함된 데이터를 기반으로 요청, 서비스 및 엣지를 필터링하는 필터 표현식 언어를 제공합니다.

X-Ray 콘솔에서 볼 트레이스의 기간을 선택할 때 콘솔에서 표시할 수 있는 결과보다 더 많은 결과를 얻을 수 있습니다. 우측 상단 모서리에서 콘솔은 스캔한 트레이스의 수 및 사용 가능한 트레이스가 더 있는지를 표시합니다. 필터 표현식을 사용하면 찾고자 하는 트레이스로 결과 범위를 좁힐 수 있습니다.

주제

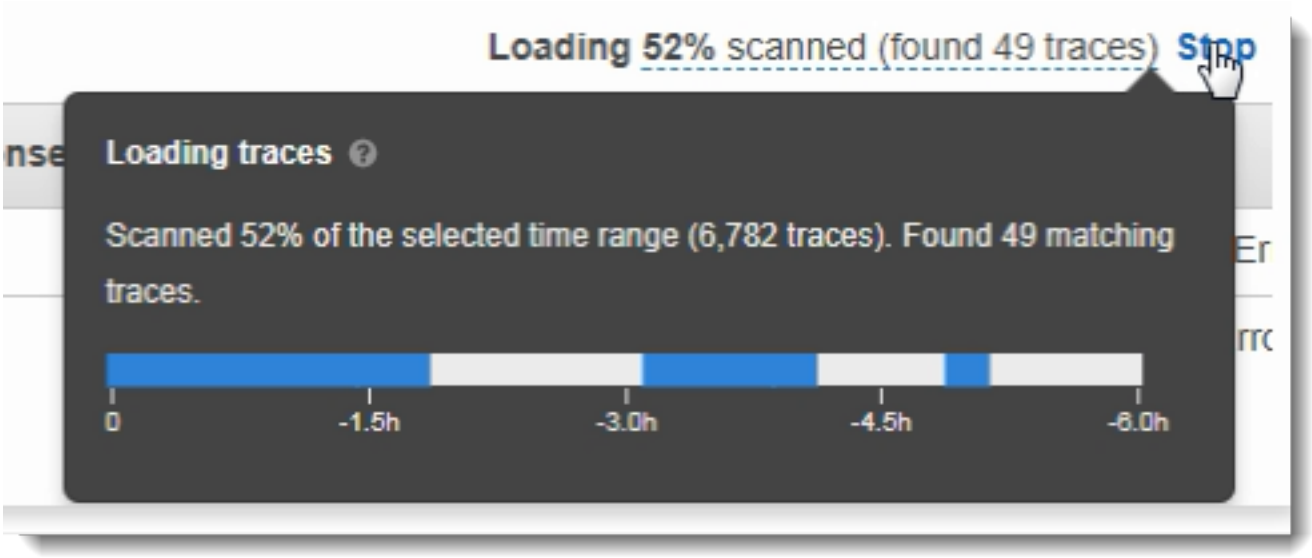
필터 표현식 세부 정보

트레이스 맵에서 노드를 선택하면 콘솔이 해당 노드의 서비스 이름 및 사용자의 선택에 따라 다르게 나타나는 오류 유형을 기반으로 필터 표현식을 생성합니다. 성능 문제를 표시하거나 특정 요청과 관련된 트레이스를 찾으려면 콘솔에서 제공하는 표현식을 조정하거나 자체적으로 표현식을 생성합니다. X-Ray SDK로 주석을 추가하는 경우 주석 키의 존재 또는 키 값을 기준으로 필터링할 수도 있습니다.

> **Note**
>
> 트레이스 맵에서 상대적 시간 범위를 선택하고 노드를 선택하면 콘솔에서 시간 범위를 절대 시작 및 종료 시간으로 변환합니다. 노드에 대한 트레이스가 검색 결과에 나타나도록 하고 노드가 활성 상태가 아닐 때 시간을 스캔하지 않도록 하기 위해 시간 범위에 노드가 트레이스를 보냈을 때 시간만 포함됩니다. 현재 시간에 상대적으로 검색하려는 경우 트레이스 페이지의 상대적 시간 범위로 다시 전환한 후 다시 스캔할 수 있습니다.

콘솔에서 표시할 수 있는 것보다 더 많은 결과가 계속 제공되는 경우 콘솔에서 일치한 트레이스 수와 스캔한 트레이스 수를 표시합니다. 표시된 백분율은 스캔한 선택된 시간 프레임의 백분율입니다. 결과에 나타난 일치하는 트레이스를 모두 보려면 필터 표현식의 범위를 더 좁히거나 보다 짧은 시간 범위를 선택합니다.

가장 최근 결과를 먼저 얻으려면 콘솔이 시간 범위의 끝에서 스캔을 시작하고 역방향으로 진행합니다. 트레이스 개수가 매우 많은데 결과가 거의 없는 경우 콘솔은 시간 범위를 청크로 분할하고 해당 청크를 병렬로 스캔합니다. 진행률 표시줄에서 스캔한 시간 범위 부분을 표시합니다.



그룹에 필터 표현식 사용

그룹은 필터 표현식으로 정의한 추적 모음입니다. 그룹을 사용하여 추가 서비스 그래프를 작성하여 Amazon CloudWatch 지표를 공급할 수 있습니다.

그룹은 이름 또는 Amazon 리소스 이름(ARN)으로 식별되며 필터 표현식을 포함합니다. 이 서비스는 수신 트레이스를 표현식과 비교하여 그에 따라 저장합니다.

필터 표현식 검색 창 왼쪽에 있는 드롭다운 메뉴를 사용하여 그룹을 생성하고 수정할 수 있습니다.

> **Note**
>
> 서비스가 그룹을 규정하는 데 오류가 발생하면 해당 그룹은 더 이상 수신 트레이스 처리에 포함되지 않으며 오류 지표가 기록됩니다.

그룹에 대한 자세한 정보는 그룹 구성 섹션을 참조하세요.

필터 표현식 구문

필터 표현식은 키워드, 유너리 또는 바이너리 연산자, 비교를 위한 값을 포함할 수 있습니다.

```
keyword operator value
```

다양한 유형의 키워드에 대해 다양한 연산자를 사용할 수 있습니다. 예를 들어 `responsetime`은 숫자 키워드이며 숫자와 관련된 연산자를 사용하여 비교할 수 있습니다.

Example – 응답 시간이 5초를 초과한 요청

```
responsetime > 5
```

AND 및 OR 연산자를 사용하여 여러 표현식을 하나의 복합 표현식으로 결합할 수 있습니다.

Example – 총 기간이 5~8초인 요청

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

단순한 키워드 및 연산자를 사용하면 트레이스 수준에서만 문제가 발견됩니다. 오류가 다운스트림에서 발생하지만 애플리케이션에 의해 처리되고 사용자에게 반환되지 않는 경우 `error` 검색에서는 발견되지 않습니다.

다운스트림 문제가 있는 트레이스를 찾기 위해 복합 키워드 `service()`와 `edge()`를 사용할 수 있습니다. 이러한 키워드를 사용하면 필터 표현식을 모든 다운스트림 노드, 단일 다운스트림 노드 또는 두 노드 간 에지에 적용할 수 있습니다. 더욱 세분화하려면 id() 함수를 사용하여 서비스와 엣지를 필터링할 수 있습니다.

부울 키워드

부울 키워드 값은 true 또는 false입니다. 오류가 발생한 트레이스를 찾으려면 이 키워드를 사용합니다.

부울 키워드

- ok – 응답 상태 코드가 2XX Success.
- `error` – 응답 상태 코드가 4XX Client Error.
- `throttle` – 응답 상태 코드가 429 요청 과다.
- `fault` – 응답 상태 코드가 5XX Server Error.
- `partial` – 요청에 완료되지 않은 세그먼트가 있음.
- `inferred` – 요청에 추론된 세그먼트가 있음
- `first` – 요소가 열거된 목록의 첫 번째임
- `last` – 요소가 열거된 목록의 마지막임
- `remote` – 근본 원인 개체가 원격임
- `root` – 서비스가 트레이스의 진입점 또는 루트 세그먼트임

부울 연산자는 지정된 키가 `true` 또는 `false`인 세그먼트를 찾습니다.

부울 연산

- none – 키워드가 true이면 표현식이 true입니다.
- ! – 키워드가 false이면 표현식이 true입니다.
- =,!= – 키워드의 값을 문자열 `true` 또는 `false`와 비교합니다. 이러한 연산자는 다른 연산자와 동일하게 작동하지만 보다 명시적입니다.

Example – 응답 상태가 2XX OK

```
ok
```

Example – 응답 상태가 2XX OK가 아님

```
!ok
```

Example – 응답 상태가 2XX OK가 아님

```
ok = false
```

Example – 마지막에 열거된 결함 트레이스의 오류 이름이 “deserialize”임

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

Example – 범위가 0.7보다 크고 서비스 이름이 “traces"인 원격 세그먼트가 포함된 요청

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

Example – 서비스 유형이 "AWS:DynamonDB"인 추론 세그먼트가 포함된 요청

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

Example – 이름이 “data-plane”인 세그먼트가 루트로 포함된 요청

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

숫자 키워드

숫자 키워드를 사용하면 특정 응답 시간, 기간 또는 응답 상태를 갖는 요청을 찾을 수 있습니다.

숫자 키워드

- `responsetime` – 서버가 응답을 전송하는 데 걸린 시간.
- `duration` – 모든 다운스트림 호출을 포함한 총 요청 기간입니다.
- `http.status` – 응답 상태 코드.
- `index` – 열거 목록에서 요소의 위치
- `coverage` – 루트 세그먼트 응답 시간 경과에 따른 개체 응답 시간의 십진 비율 응답 시간 근본 원인 개체일 때만 사용 가능합니다.

숫자 연산자

숫자 키워드는 표준 등식 및 비교 연산자를 사용합니다.

- =,!= – 이 키워드는 숫자값과 같음 또는 같지 않음입니다.
- <,<=, >,>= – 이 키워드는 숫자값보다 작음 또는 큼입니다.

Example – 응답 상태가 200 OK가 아님

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

Example – 총 기간이 5~8초인 요청

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

Example – 모든 다운스트림 호출을 포함하여 3초 이내에 성공적으로 완료된 요청

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

Example – 5보다 큰 인덱스가 포함된 열거된 목록 개체

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

Example – 0.8보다 큰 범위가 포함된 마지막 개체인 요청

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

문자열 키워드

문자열 키워드를 사용하면 요청 헤더 또는 특정 사용자 ID에 특정 텍스트가 포함된 트레이스를 찾을 수 있습니다.

문자열 키워드

- `http.url` – 요청 URL.
- `http.method` – 요청 메서드.
- `http.useragent` – 요청 사용자 에이전트 문자열.
- `http.clientip` – 요청자 IP 주소.
- `user` – 트레이스 내 세그먼트의 사용자 필드 값.
- `name` – 서비스 또는 예외 이름
- `type` – 서비스 유형
- `message` – 예외 메시지
- `availabilityzone` – 트레이스 내 세그먼트의 availabilityzone 필드 값
- `instance.id` – 트레이스 내 세그먼트의 인스턴스 ID 필드 값
- `resource.arn` – 트레이스 내 세그먼트의 리소스 ARN 필드 값

문자열 연산자는 특정 텍스트와 같거나 이를 포함하는 값을 찾습니다. 값은 항상 인용 부호를 사용하여 지정해야 합니다.

문자열 연산자

- `=`,`!=` – 이 키워드는 숫자값과 같음 또는 같지 않음입니다.
- `CONTAINS` – 이 키워드는 특정 문자열을 포함합니다.
- `BEGINSWITH` , `ENDSWITH` – 이 키워드는 특정 문자열로 시작하거나 끝납니다.

Example – http.url 필터

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

트레이스에서 값과 상관없이 특정 필드가 존재하는지 테스트하려면 필드가 빈 문자열을 포함하는지 확인합니다.

Example – 사용자 필터

사용자 ID가 있는 모든 트레이스를 찾습니다.

```
user CONTAINS ""
```

Example – “Auth”라는 이름의 서비스가 포함된 결함 근본 원인이 있는 트레이스를 선택합니다.

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

Example – 마지막 서비스에 DynamoDB 유형이 포함된 응답 시간 근본 원인이 있는 트레이스를 선택합니다.

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

Example – 마지막 예외에 "access denied for account_id: 1234567890" 메시지가 포함된 결함 근본 원인이 있는 추적을 선택합니다.

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

복합 키워드

복합 키워드를 사용하여 서비스 이름, 엣지 이름 또는 주석 값을 기준으로 요청을 찾을 수 있습니다. 서비스 및 엣지의 경우 서비스 또는 엣지에 적용되는 추가 필터 표현식을 지정할 수 있습니다. 주석의 경우 부울, 숫자 또는 문자열 연산자를 사용하여 특정 키로 주석 값을 필터링할 수 있습니다.

복합 키워드

- `annotation[`*`key`*`]` – *`key`* 필드가 있는 주석의 값. 주석 값은 부울, 숫자 또는 문자열일 수 있으므로 이러한 유형의 비교 연산자를 모두 사용할 수 있습니다. 이 키워드는 `service` 또는 `edge` 키워드와 함께 사용할 수 있습니다. 점(마침표)이 포함된 주석 키는 대괄호([ ])로 묶어야 합니다.
- `edge(`*`source`*`, `*`destination`*`) {`*`filter`*`}` – *`source`* 서비스와 *`destination`* 서비스 사이를 연결합니다. 선택적으로, 이 연결의 세그먼트에 적용되는 필터 표현식을 중괄호로 묶을 수 있습니다.

- `group.name / group.arn` – 그룹 이름 또는 그룹 ARN으로 참조되는 그룹의 필터 표현식 값입니다.
- `json` – JSON 근본 원인 객체. 프로그래밍 방식으로 JSON 개체를 생성하는 단계는 AWS X-Ray에서 데이터 가져오기를 참조하세요.
- `service(name) {filter}` – 이름이 *name*인 서비스. 선택적으로, 서비스에서 생성하는 세그먼트에 적용되는 필터 표현식을 중괄호로 묶을 수 있습니다.

트레이스 맵에서 특정 노드에 도달한 요청의 트레이스를 찾으려면 서비스 키워드를 사용합니다.

복합 키워드 연산자는 지정된 키가 설정되었거나 설정되지 않은 세그먼트를 찾습니다.

복합 키워드 연산자

- none – 키워드가 설정된 경우 표현식이 true입니다. 키워드가 부울 유형인 경우 부울 값으로 평가됩니다.
- `!` – 키워드가 설정되지 않은 경우 표현식이 true입니다. 키워드가 부울 유형인 경우 부울 값으로 평가됩니다.
- =,!= - 키워드의 값을 비교합니다.
- `edge(source, destination) {filter}` – *source* 서비스와 *destination* 서비스 사이를 연결합니다. 선택적으로, 이 연결의 세그먼트에 적용되는 필터 표현식을 중괄호로 묶을 수 있습니다.
- `annotation[key]` – *key* 필드가 있는 주석의 값. 주석 값은 부울, 숫자 또는 문자열일 수 있으므로 이러한 유형의 비교 연산자를 모두 사용할 수 있습니다. 이 키워드는 `service` 또는 `edge` 키워드와 함께 사용할 수 있습니다.
- `json` – JSON 근본 원인 객체. 프로그래밍 방식으로 JSON 개체를 생성하는 단계는 AWS X-Ray에서 데이터 가져오기를 참조하세요.

트레이스 맵에서 특정 노드에 도달한 요청의 트레이스를 찾으려면 서비스 키워드를 사용합니다.

Example – 서비스 필터

`api.example.com` 호출을 포함하고 오류(500 시리즈 오류)가 발생한 요청

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

서비스 이름을 제외하여 필터 표현식을 서비스 맵의 모든 노드에 적용할 수 있습니다.

Example – 서비스 필터

트레이스 맵의 어느 곳에서든 오류를 일으킨 요청입니다.

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

에지 키워드는 필터 표현식을 두 노드 간 연결에 적용합니다.

Example – 엣지 필터

`api.example.com` 서비스가 `backend.example.com`을 호출했으나 실패하고 오류가 발생한 요청

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

또한 서비스 및 에지 키워드와 함께 `!` 연산자를 사용하여 다른 필터 표현식의 결과에서 서비스 또는 에지를 제외할 수 있습니다.

Example – 서비스 및 요청 필터

URL이 `http://api.example.com/`으로 시작하고 `/v2/`를 포함하지만 `api.example.com` 서비스에 연결되지 않는 요청

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

Example – 서비스 및 응답 시간 필터

`http url`이 설정되어 있고 응답 시간이 2초보다 큰 트레이스를 찾습니다.

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

주석의 경우 `annotation[`*key*`]`가 설정된 모든 트레이스를 직접 호출하거나 값 유형에 해당하는 비교 연산자를 사용할 수 있습니다.

Example – 문자열 값을 포함하는 주석

문자열 값이 `gameid`인 `"817DL6V0"`라는 주석이 포함된 요청

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

Example – 주석이 설정되었습니다.

age set이라는 주석이 포함된 요청.

```
annotation[age]
```

Example – 주석이 설정되지 않았습니다.

age set이라는 주석이 없는 요청.

```
!annotation[age]
```

Example – 숫자 값을 포함하는 주석

주석의 수명이 숫자 값 29보다 큰 요청

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

Example – 서비스 또는 엣지와 조합한 주석

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

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

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

Example – 사용자가 있는 그룹

트레이스가 high_response_time 그룹 필터(예:responseTime > 3)를 충족하고 사용자 이름이 Alice인 요청.

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

Example – 근본 원인 개체가 포함된 JSON

일치하는 근본 원인 개체가 포함된 요청

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

id 함수

`service` 또는 `edge` 키워드에 서비스 이름을 제공하면 해당 이름의 모든 노드가 결과로 반환됩니다. 보다 세밀하게 필터링하려면 `id` 함수로 이름 외에 서비스 유형도 지정하여 이름이 같은 노드를 구별할 수 있습니다.

모니터링 계정에서 여러 계정의 트레이스를 볼 때 이 `account.id` 함수를 사용하여 서비스의 특정 계정을 지정합니다.

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

서비스 및 엣지 필터에 서비스 이름 대신 `id` 함수를 사용할 수 있습니다.

```
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 }
```

예를 들어 AWS Lambda 함수는 트레이스 맵에 두 개의 노드를 생성합니다. 하나는 함수 호출용이고 다른 하나는 Lambda 서비스용입니다. 두 노드는 이름이 같지만 유형이 다릅니다. 표준 서비스 필터는 두 노드 모두의 트레이스를 찾습니다.

Example – 서비스 필터

`random-name`이라는 서비스에 대한 오류를 포함하는 요청

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

서비스로 인한 오류를 제외하고 함수 자체의 오류로 검색 범위를 좁히려면 `id` 함수를 사용합니다.

Example – id 함수를 사용한 서비스 필터

유형이 `random-name`인 `AWS::Lambda::Function` 서비스의 오류를 포함하는 요청

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

또한 이름을 완전히 제외하여 노드를 유형별로 검색할 수 있습니다.

Example – id 기능 및 서비스 유형을 포함하는 서비스 필터

유형이 `AWS::Lambda::Function`인 서비스의 오류를 포함하는 요청

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

특정 노드를 검색하려면 계정 ID를 AWS 계정지정합니다.

Example – id 기능 및 계정 ID를 사용한 서비스 필터

특정 계정 ID `AWS::Lambda::Function` 내에 서비스를 포함하는 요청.

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

교차 계정 추적

AWS X-Ray 는 교차 계정 관찰성을 지원하므로 내의 여러 계정에 걸쳐 있는 애플리케이션을 모니터링하고 문제를 해결할 수 있습니다 AWS 리전. 연결된 모든 계정의 지표, 로그 및 추적을 마치 하나의 계정에서 작업하는 것처럼 원활하게 검색, 시각화 및 분석할 수 있습니다. 이를 통해 여러 계정에 걸쳐 발생하는 요청을 전체적으로 볼 수 있습니다. CloudWatch 콘솔 내의 X-Ray 트레이스 맵과 트레이스 페이지에서 교차 계정 추적을 확인할 수 있습니다.

공유된 통합 관찰성 데이터에는 다음 유형의 원격 분석이 포함될 수 있습니다.

- Amazon CloudWatch의 지표
- Amazon CloudWatch Logs의 로그 그룹
- 의 트레이스 AWS X-Ray
- Amazon CloudWatch Application Insights의 애플리케이션

교차 계정 관찰성 구성

교차 계정 관찰성을 켜려면 하나 이상의 AWS 모니터링 계정을 설정하고 여러 소스 계정과 연결합니다. 모니터링 계정은 소스 계정에서 생성된 관찰성 데이터를 보고 상호 작용할 AWS 계정 수 있는 중앙 계정입니다. 소스 계정은 포함된 리소스에 대한 관찰성 데이터를 AWS 계정 생성하는 개인입니다.

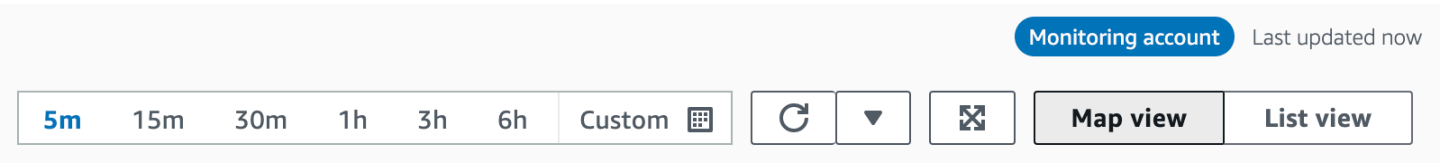
소스 계정은 관찰성 데이터를 모니터링 계정과 공유합니다. 트레이스는 각 소스 계정에서 최대 5개의 모니터링 계정으로 복사됩니다. 소스 계정에서 첫 번째 모니터링 계정으로의 트레이스 사본은 무료입니다. 추가 모니터링 계정으로 전송된 트레이스 사본은 표준 요금에 따라 각 소스 계정에 청구됩니다. 요금에 대한 자세한 정보는 AWS X-Ray 요금 및 Amazon CloudWatch 요금을 참조하세요.

모니터링 계정과 소스 계정 간에 링크를 생성하려면 CloudWatch 콘솔 또는 AWS CLI 및 API의 새 Observability Access Manager 명령을 사용합니다. 자세한 내용은 CloudWatch 교차 계정 관찰성을 참조하세요.

> **Note**
>
> X-Ray 추적은 수신 AWS 계정 된에 청구됩니다. 샘플링된 요청이 둘 이상의 서비스에 걸쳐 있는 경우 AWS 계정각 계정은 별도의 트레이스를 기록하고 모든 트레이스는 동일한 트레이스 ID를 공유합니다. 계정 간 관찰성 요금에 대한 자세한 내용은 AWS X-Ray 요금 및 Amazon CloudWatch 요금을 참조하십시오.

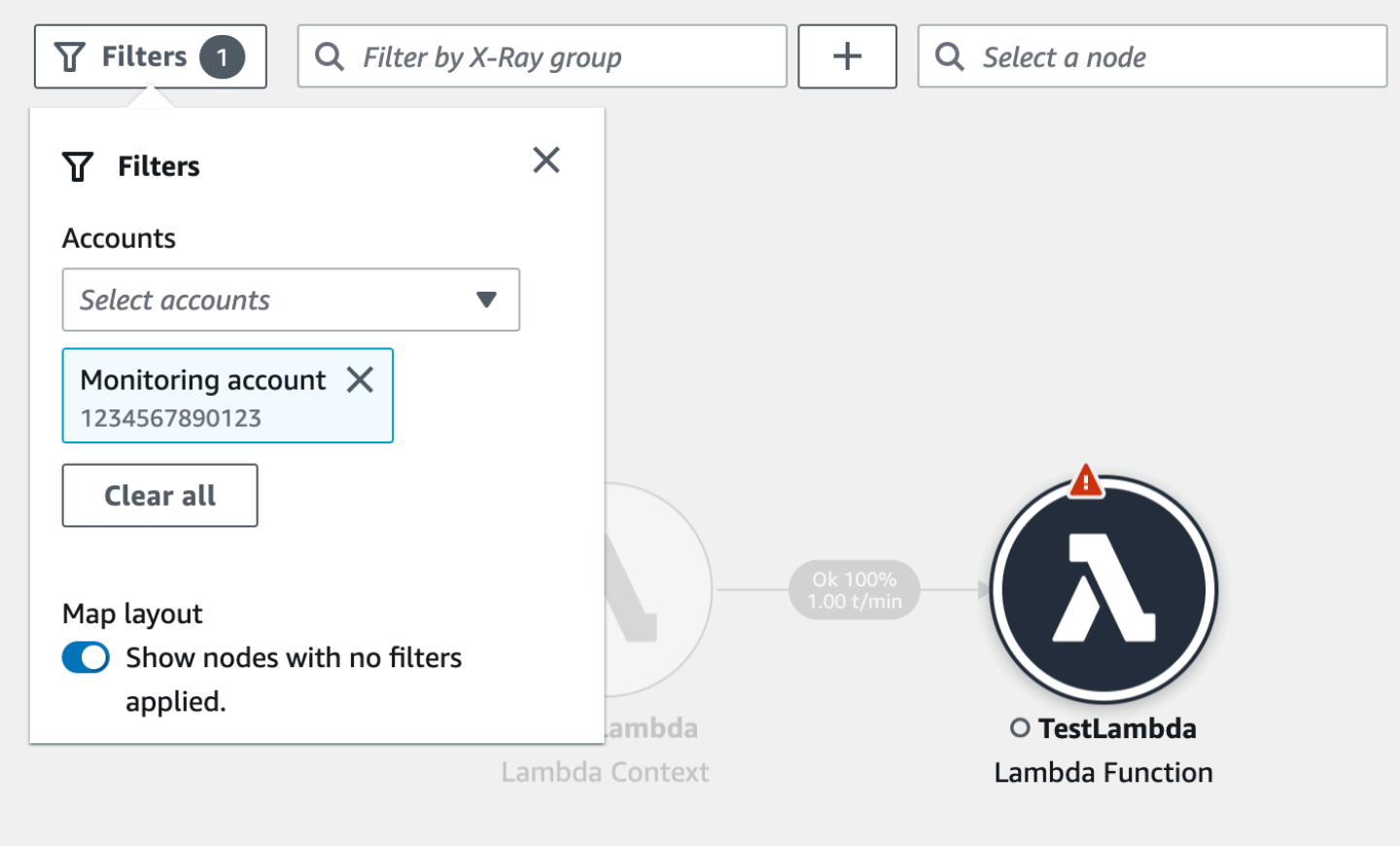
교차 계정 추적 보기

교차 계정 추적은 모니터링 계정에 표시됩니다. 각 소스 계정에는 해당 특정 계정의 로컬 트레이스만 표시됩니다. 다음 섹션에서는 모니터링 계정에 로그인하고 Amazon CloudWatch 콘솔을 열었다고 가정합니다. 트레이스 맵과 트레이스 페이지 모두에서 모니터링 계정 배지가 오른쪽 상단에 표시됩니다.

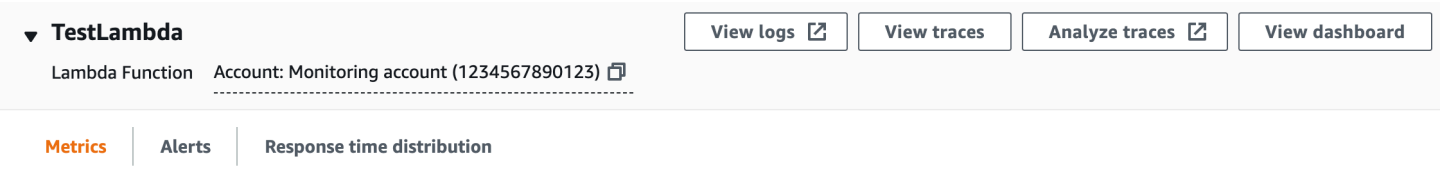


트레이스 맵

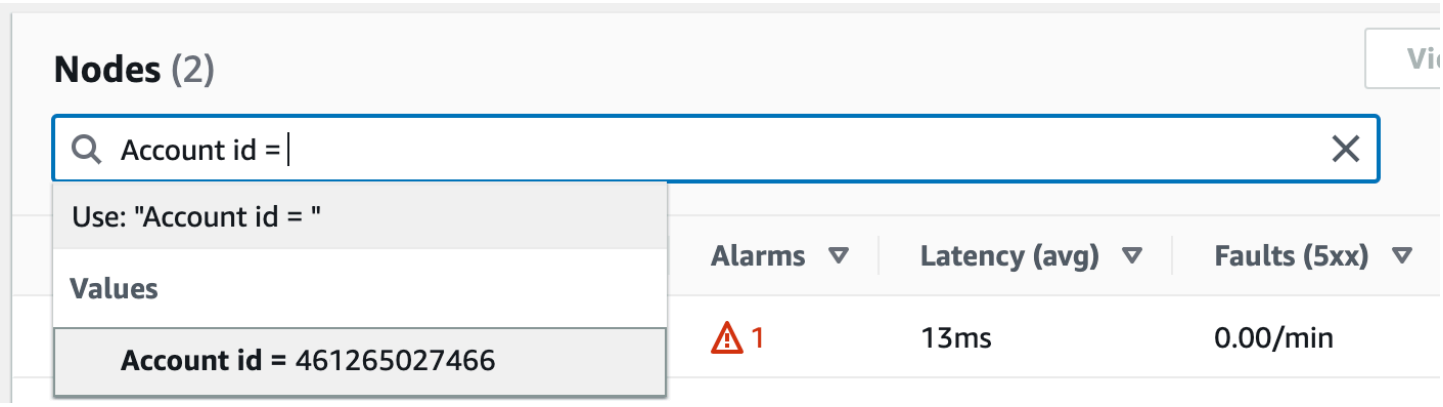
CloudWatch 콘솔의 왼쪽 탐색 창에 있는 X-Ray 트레이스에서 트레이스 맵을 선택합니다. 기본적으로 트레이스 맵에는 모니터링 계정에 트레이스를 보내는 모든 소스 계정에 대한 노드와 모니터링 계정 자체에 대한 노드가 표시됩니다. 트레이스 맵의 왼쪽 상단에 있는 필터를 선택하여 계정 드롭다운을 사용하여 트레이스 맵을 필터링합니다. 계정 필터가 적용되면 현재 필터와 일치하지 않는 계정의 서비스 노드는 회색으로 표시됩니다.



서비스 노드를 선택하면 노드 세부정보 창에 서비스의 계정 ID와 레이블이 포함됩니다.



트레이스 맵의 오른쪽 상단에서 목록 보기를 선택하면 서비스 노드 목록을 확인할 수 있습니다. 서비스 노드 목록에는 모니터링 계정 및 구성된 모든 소스 계정의 서비스가 포함됩니다. 노드 필터에서 선택하여 계정 레이블 또는 계정 ID를 기준으로 노드 목록을 필터링합니다.

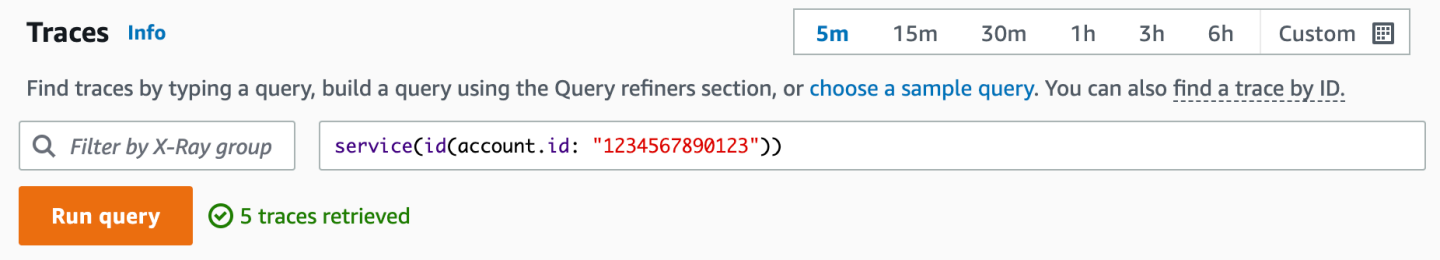


트레이스

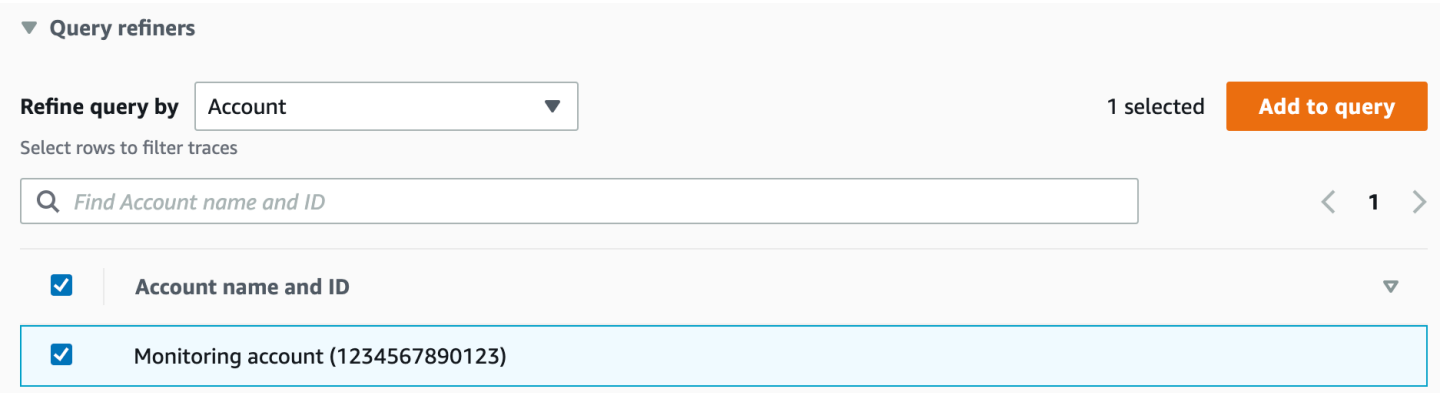
모니터링 계정에서 CloudWatch 콘솔을 열고 왼쪽 탐색 창의 X-Ray traces에서 Traces를 선택하여 여러 계정에 걸친 추적의 세부 정보를 확인합니다. X-Ray 트레이스 맵에서 노드를 선택한 다음 노드 세부 정보 창에서 트레이스 보기를 선택하여 이 페이지를 열 수도 있습니다.

Traces(추적) 페이지는 계정 ID로 쿼리를 지원합니다. 시작하려면 하나 이상의 계정 ID가 포함된 쿼리를 입력합니다. 다음 예시에서는 계정 ID X 또는 Y를 통과한 트레이스를 쿼리합니다.

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



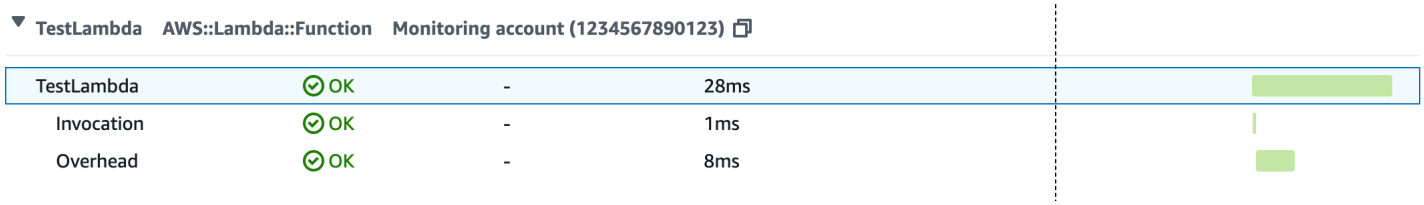
계정별로 쿼리를 구체화하세요. 목록에서 하나 이상의 계정을 선택하고 Add to query(쿼리에 추가)를 선택합니다.



트레이스 세부 정보

트레이스 페이지 하단의 트레이스 목록에서 트레이스를 선택하여 트레이스의 세부 정보를 볼 수 있습니다. 트레이스 세부 정보에는 트레이스가 통과한 모든 계정의 서비스 노드를 포함한 트레이스 세부 정보 맵이 표시됩니다. 특정 서비스 노드를 선택하면 해당 계정을 확인할 수 있습니다.

세그먼트 타임라인 섹션에는 타임라인의 각 세그먼트에 대한 계정 세부 정보가 표시됩니다.



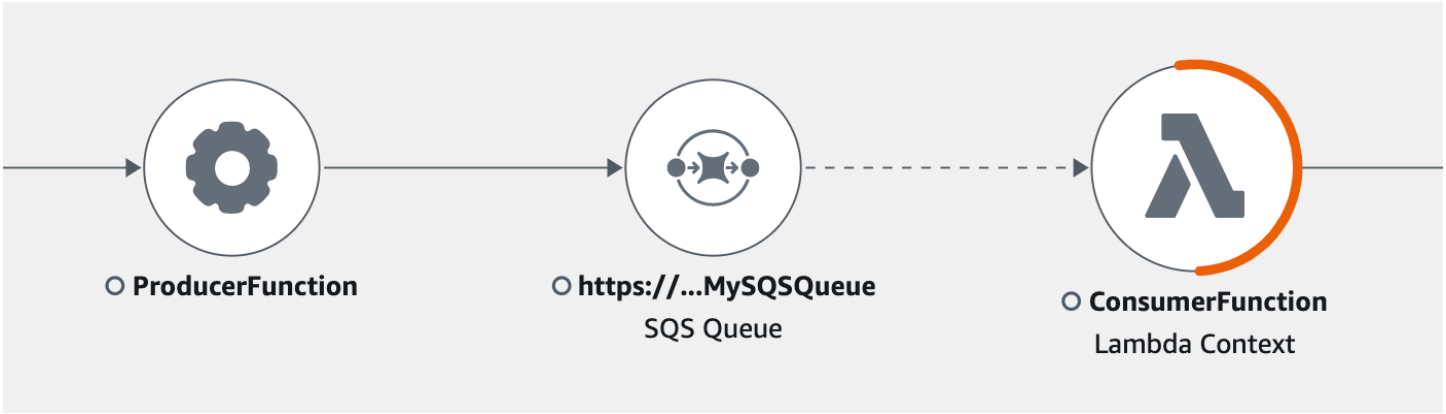
이벤트 기반 애플리케이션 추적

AWS X-Ray 는 Amazon SQS 및를 사용하여 이벤트 기반 애플리케이션 추적을 지원합니다 AWS Lambda. CloudWatch 콘솔을 사용하면 각 요청이 Amazon SQS로 대기열에 추가되고 하나 이상의 Lambda 함수에 의해 처리될 때 연결된 보기를 볼 수 있습니다. 업스트림 메시지 생산자의 트레이스가 다운스트림 Lambda 함수의 트레이스에 자동으로 연결되므로 전체 애플리케이션을 종합적으로 파악할 수 있습니다.

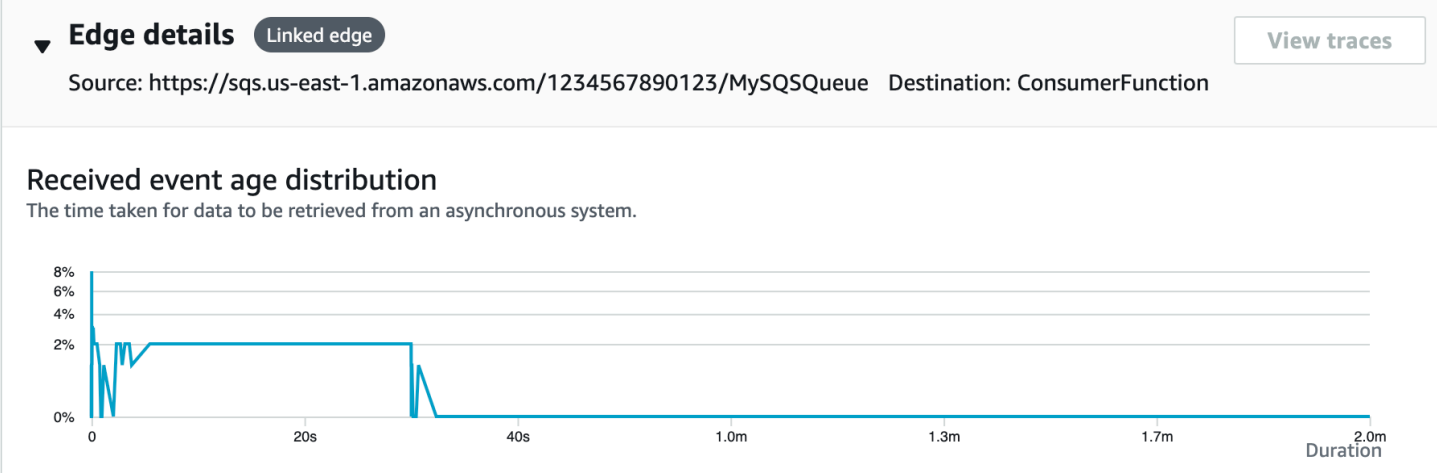
> **Note**
>
> 각 추적 세그먼트는 최대 20개의 추적에 연결할 수 있으며, 추적에는 최대 100개의 링크가 포함될 수 있습니다. 특정 시나리오에서 추가 추적을 연결하면 최대 추적 문서 크기를 초과하여 잠재적으로 불완전한 추적을 초래할 수 있습니다. 예를 들어 추적이 활성화된 Lambda 함수가 한 번의 간접 호출로 많은 SQS 메시지를 대기열에 전송하는 경우 이런 일이 발생할 수 있습니다. 이 문제가 발생하는 경우 X-Ray SDK를 사용하는 완화 방법을 사용할 수 있습니다. 자세한 내용은 자바, Node.js, Python, Go 또는 .NET용 X-Ray SDK를 참조하십시오.

트레이스 맵에서 연결된 트레이스 보기

CloudWatch 콘솔 내의 트레이스 맵 페이지에서는 메시지 생산자의 트레이스가 Lambda 소비자의 트레이스에 연결된 트레이스 맵을 볼 수 있습니다. 이러한 링크는 Amazon SQS 노드와 다운스트림 Lambda 소비자 노드를 연결하는 점선 엣지와 함께 표시됩니다.



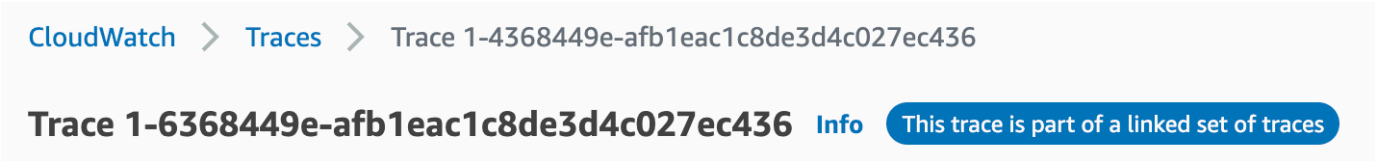
점선 엣지를 선택하면 수신된 이벤트 기간 히스토그램을 표시합니다. 이 히스토그램은 소비자가 수신한 이벤트 기간의 분포를 매핑합니다. 기간은 이벤트가 수신될 때마다 계산됩니다.



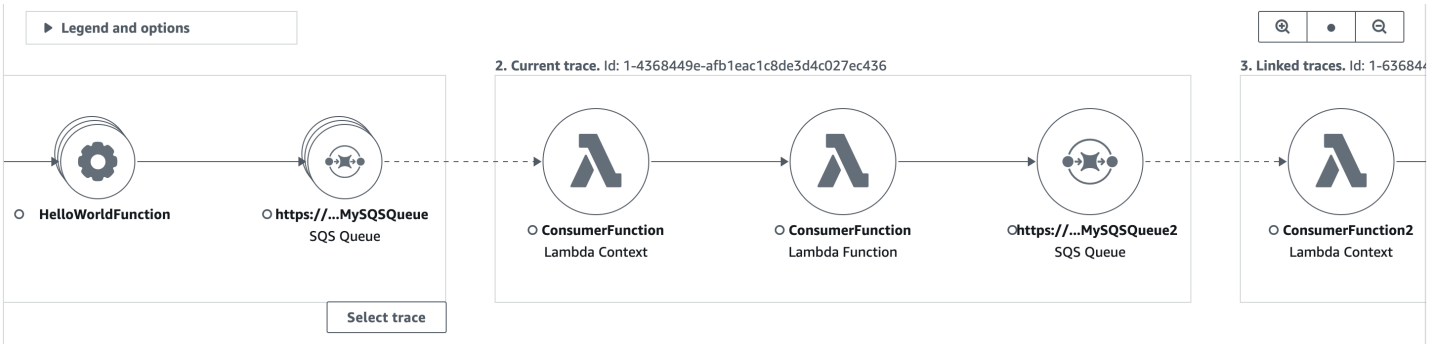
연결된 추적 세부 정보 보기

메시지 생산자, Amazon SQS 대기열 또는 람다 소비자로부터 전송된 추적 세부 정보를 확인합니다:

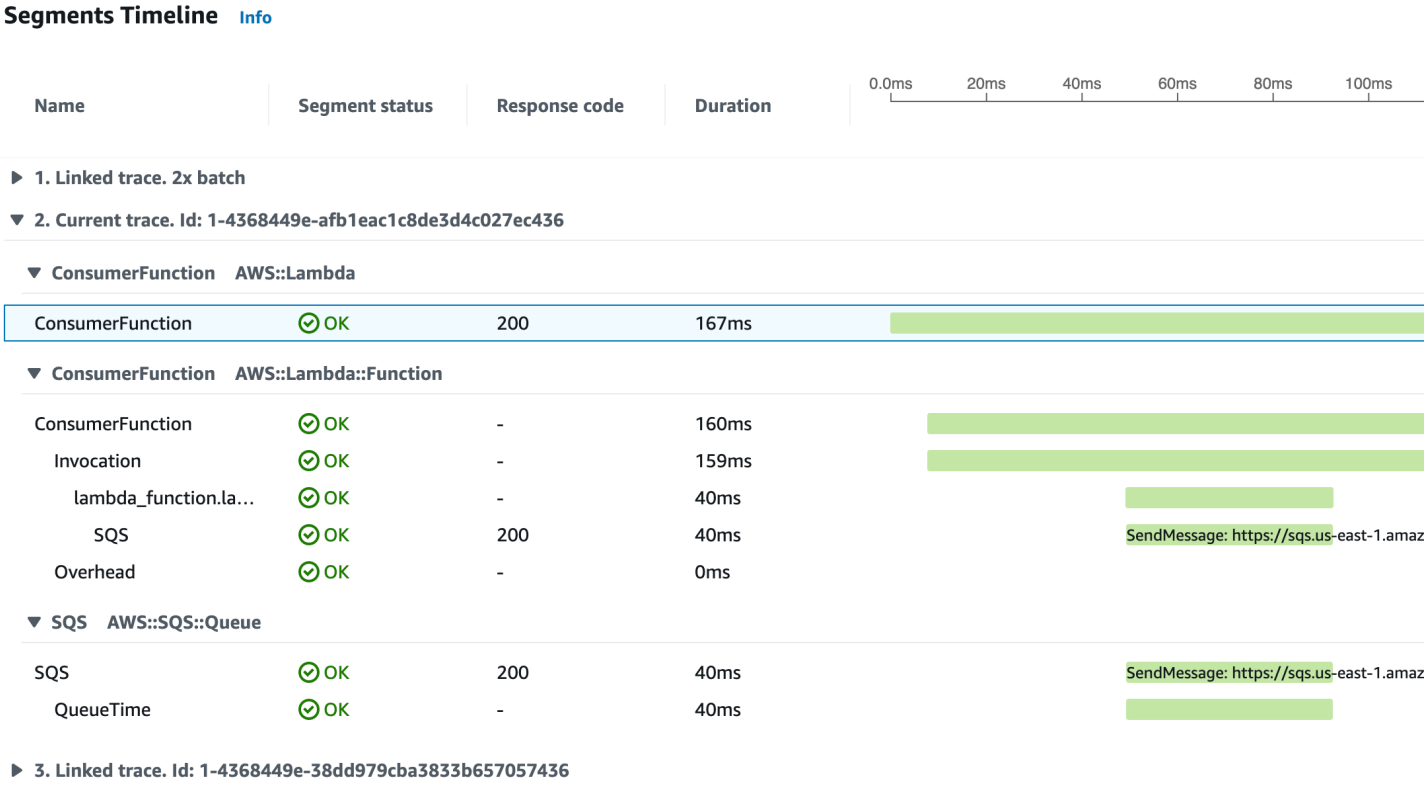
1. 트레이스 맵을 사용하여 메시지 생산자, Amazon SQS 또는 Lambda 소비자 노드를 선택합니다.
2. 노드 세부 정보 창에서 추적 보기를 선택하여 추적 목록을 표시합니다. CloudWatch 콘솔 내에서 추적 페이지로 직접 이동할 수도 있습니다.
3. 목록에서 특정 추적을 선택하면 추적 세부정보 페이지가 열립니다. 선택한 추적이 연결된 추적 집합의 일부인 경우 추적 세부정보 페이지에 메시지가 표시됩니다.



트레이스 세부 정보 맵에는 현재 트레이스와 함께 업스트림 및 다운스트림에 연결된 트레이스가 표시되며, 각 트레이스의 경계를 나타내는 상자 안에 각 트레이스가 포함되어 있습니다. 현재 선택된 트레이스가 여러 업스트림 또는 다운스트림 트레이스에 연결되어 있는 경우, 연결된 업스트림 또는 다운스트림 트레이스 내의 노드가 스택되고 트레이스 선택 버튼이 표시됩니다.



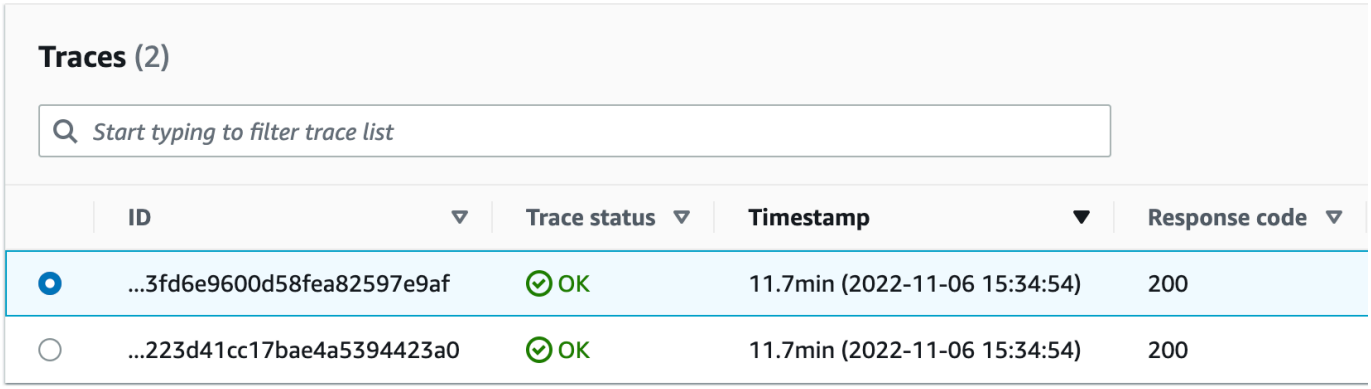
트레이스 세부 정보 맵 아래에는 업스트림 및 다운스트림에 연결된 트레이스가 포함된 트레이스 세그먼트의 타임라인이 표시됩니다. 업스트림 또는 다운스트림에 연결된 트레이스가 여러 개 있는 경우 해당 세그먼트 세부 정보를 표시할 수 없습니다. 연결된 추적 세트 내의 단일 추적에 대한 세그먼트 세부 정보를 보려면 아래에 설명된 대로 단일 추적을 선택합니다.



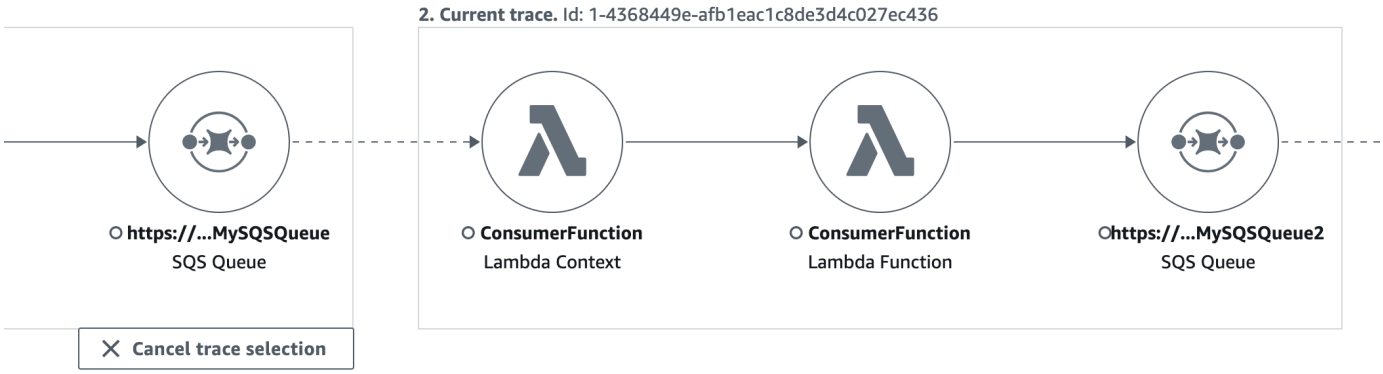
연결된 트레이스 세트 내에서 단일 트레이스 선택하기

연결된 트레이스 세트를 단일 트레이스에 필터링하여 타임라인에서 세그먼트 세부 정보를 확인할 수 있습니다.

1. 트레이스 세부 정보 맵의 연결된 트레이스 아래에서 트레이스 선택을 선택합니다. 추적 목록이 표시됩니다.



2. 트레이스 옆의 라디오 버튼을 선택하여 트레이스 세부 정보 맵에서 트레이스를 확인합니다.
3. 연결된 트레이스의 전체 세트를 보려면 트레이스 선택 취소를 선택합니다.



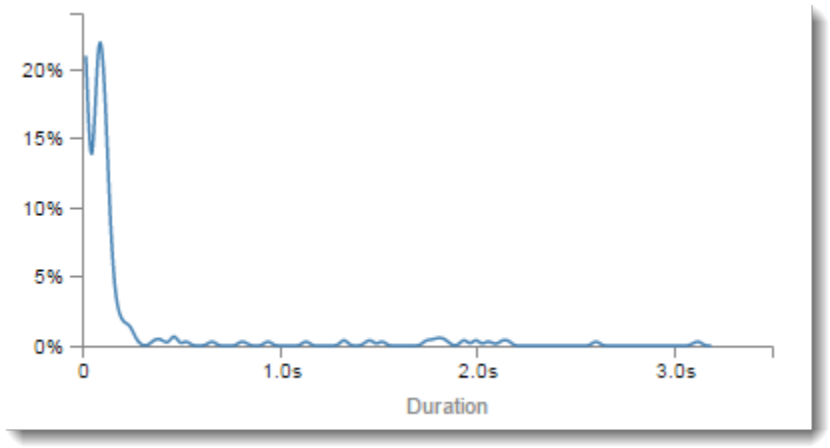
지연 시간 히스토그램 사용

AWS X-Ray 추적 맵에서 노드 또는 엣지를 선택하면 X-Ray 콘솔에 지연 시간 분포 히스토그램이 표시됩니다.

지연 시간

지연 시간은 요청 시작에서 요청 완료까지의 시간입니다. 히스토그램은 지연 시간의 분포를 보여줍니다. x축에는 기간이 표시되고, y축에는 각 기간에 해당하는 요청의 비율이 표시됩니다.

이 히스토그램은 300ms 미만에 요청이 대부분 완료되는 서비스를 보여줍니다. 소수의 요청이 최대 2초까지 걸리며, 몇몇 특이값은 이보다 오래 걸립니다.



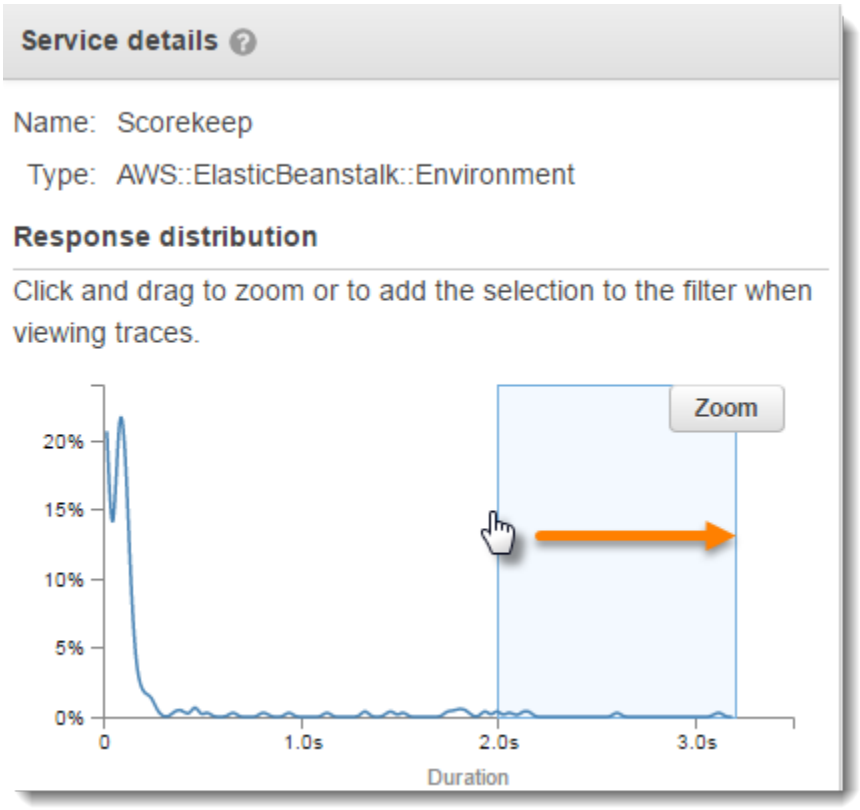
서비스 세부 정보 해석

서비스 히스토그램과 에지 히스토그램은 서비스 또는 요청자의 관점에서 지연 시간을 시각적으로 표현합니다.

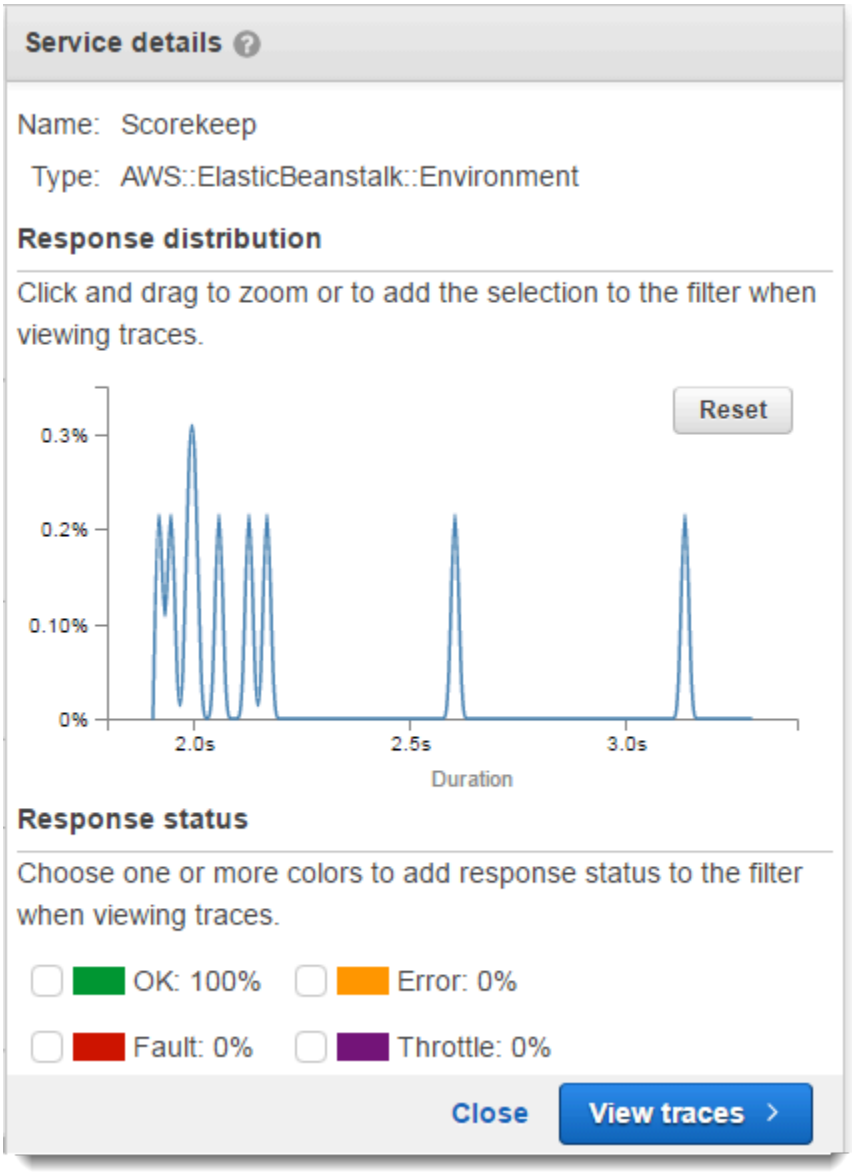
- 원을 클릭하여 서비스 노드를 선택합니다. X-Ray는 서비스에서 처리한 요청에 대한 히스토그램을 보여줍니다. 지연 시간은 서비스에 의해 기록된 것이며 서비스와 요청자 간 네트워크 지연 시간은 포함하지 않습니다.
- 두 서비스 간 가장자리의 선이나 화살표 끝을 클릭하여 엣지를 선택합니다. X-Ray는 다운스트림 서비스에서 처리한 요청자의 요청에 대한 히스토그램을 보여줍니다. 지연 시간은 요청자에 의해 기록된 것이며 두 서비스 간 네트워크 연결 지연 시간을 포함합니다.

[Service details] 패널 히스토그램을 해석하려면 히스토그램에서 대부분의 값과 다른 값을 찾아야 합니다. 이러한 특이값은 히스토그램에서 피크 또는 스파이크로 보일 수 있으며, 현재 상황을 조사하기 위해 특정 영역의 트레이스를 볼 수 있습니다.

지연 시간을 기준으로 필터링된 트레이스를 보려면 히스토그램에서 범위를 선택합니다. 선택을 시작하려는 지점을 클릭하고 왼쪽에서 오른쪽으로 끌어 트레이스 필터에 포함시킬 지연 시간 범위를 강조 표시합니다.



범위를 선택한 후 [Zoom]를 선택하여 히스토그램의 해당 부분만 보고 선택을 미세 조정할 수 있습니다.



보려는 영역에 초점을 맞췄으면 [View traces]를 선택합니다.

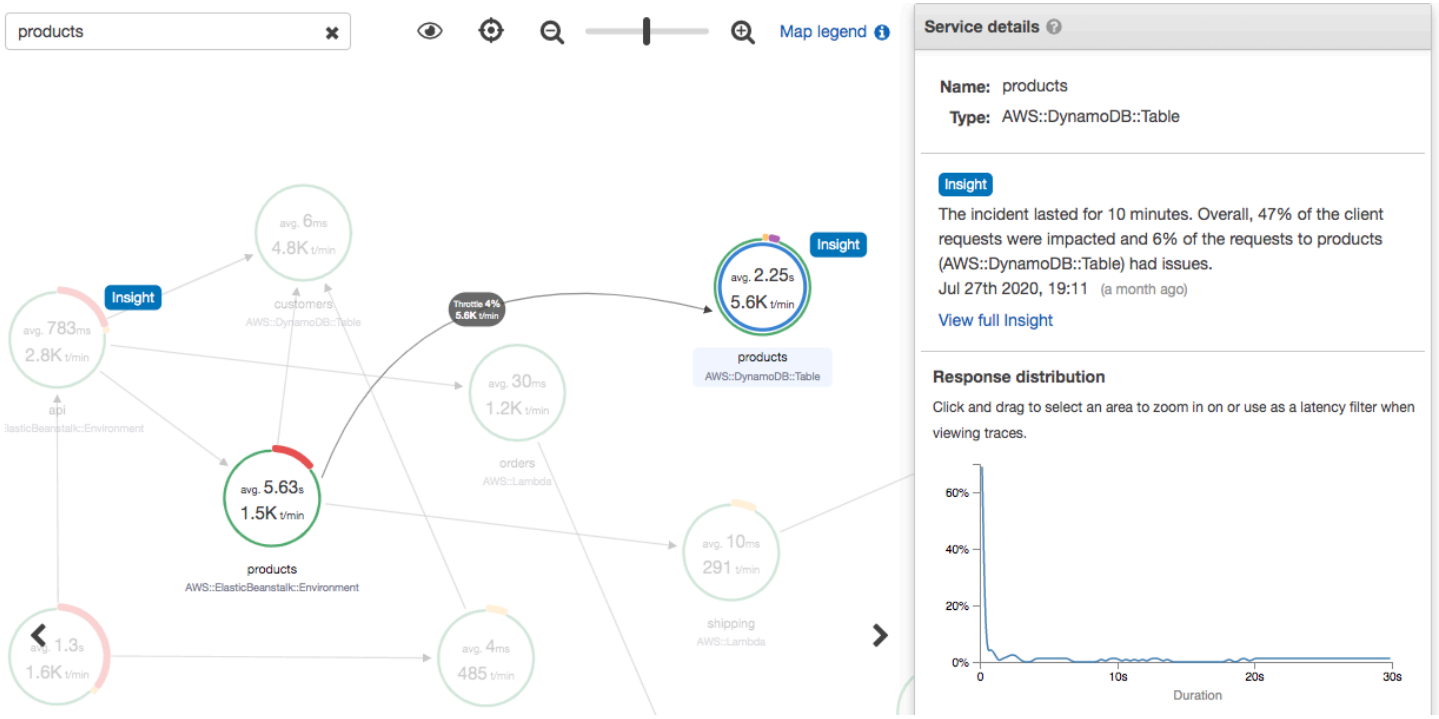
X-Ray 인사이트 사용

AWS X-Ray 는 계정의 추적 데이터를 지속적으로 분석하여 애플리케이션의 긴급한 문제를 식별합니다. 결함률이 예상 범위를 초과하면 문제를 기록하고 문제가 해결될 때까지 그 영향을 추적하는 인사이트를 생성합니다. 인사이트를 통해 다음을 수행할 수 있습니다.

- 애플리케이션에서 문제가 발생하는 위치, 문제의 근본 원인 및 관련 영향을 파악합니다. 인사이트에서 제공하는 영향 분석을 통해 이슈의 심각도와 우선순위를 도출할 수 있습니다.

- 시간이 지남에 따라 문제가 변경될 때마다 알림을 받습니다. Amazon EventBridge를 사용하여 인사이트 알림을 모니터링 및 알림 솔루션과 통합할 수 있습니다. 이 통합 기능을 사용하면 문제의 심각도에 따라 자동화된 이메일 또는 알림을 보낼 수 있습니다.

X-Ray 콘솔은 트레이스 맵에서 진행 중인 인시던트가 있는 노드를 식별합니다. 인사이트 요약을 보려면 해당 노드를 선택하십시오. 왼쪽 탐색 창에서 인사이트를 선택하여 인사이트를 확인하고 필터링할 수도 있습니다.



X-Ray는 서비스 맵에서 하나 이상의 노드에서 이상을 감지할 때 인사이트를 제공합니다. 이 서비스는 통계적 모델링을 사용하여 애플리케이션에서 예상되는 서비스 장애율을 예측합니다. 이전 예제에서 이상은 결함의 증가입니다 AWS Elastic Beanstalk. Elastic Beanstalk 서버에서 여러 번의 API 직접 호출 시간 초과 현상이 발생하여 다운스트림 노드에 이상 징후가 발생했습니다.

X-Ray 콘솔에서 인사이트 활성화하기

인사이트 기능을 사용하려는 각 그룹에서 인사이트를 활성화해야 합니다. 그룹 페이지에서 인사이트를 활성화할 수 있습니다.

1. X-Ray 콘솔을 엽니다.
2. 기존 그룹을 선택하거나 그룹 생성을 선택하여 새 그룹을 만든 다음 인사이트 활성화를 선택합니다. X-Ray 콘솔에서 그룹을 구성하는 방법에 대한 자세한 내용은 그룹 구성를 참조하세요.
3. 왼쪽의 탐색 창에서 인사이트를 선택한 다음 확인하려는 인사이트를 선택합니다.

> **Note**
>
> X-Ray는 인사이트에서 데이터를 검색하기 위해 GetInsightSummaries, GetInsight, GetInsightEvents 및 GetInsightImpactGraph API 작업을 사용합니다.
> 자세한 내용은 AWS X-Ray 에서 IAM을 사용하는 방법 단원을 참조하십시오.

인사이트 알림 활성화

인사이트 알림을 사용하면 인사이트가 생성되거나, 크게 변경되거나, 종료된 시점과 같은 각 인사이트 이벤트에 대해 알림이 생성됩니다. 고객은 Amazon EventBridge 이벤트를 통해 이러한 알림을 수신할 수 있으며, 조건부 규칙을 사용하여 SNS 알림, Lambda 간접 호출, SQS 대기열에 메시지 게시 또는 EventBridge가 지원하는 모든 대상과 같은 작업을 수행할 수 있습니다. 인사이트 알림은 최선의 노력을 다해 발송되지만 반드시 발송된다는 보장은 없습니다. 대상에 대한 자세한 내용은 Amazon EventBridge 대상을 참조하세요.

그룹 페이지에서 인사이트가 활성화된 모든 그룹에 대해 인사이트 알림을 활성화할 수 있습니다.

X-Ray 그룹에 대한 알림을 활성화하려면

1. X-Ray 콘솔을 엽니다.
2. 기존 그룹을 선택하거나 그룹 생성을 선택하여 새 그룹을 만든 다음 인사이트 활성화가 선택되어 있는지 확인한 다음, 알림 활성화를 선택합니다. X-Ray 콘솔에서 그룹을 구성하는 방법에 대한 자세한 내용은 그룹 구성를 참조하세요.

Amazon EventBridge 조건부 규칙을 구성하려면

1. EventBridge 콘솔을 엽니다.
2. 왼쪽 탐색 표시줄의 규칙으로 이동하여 규칙 생성을 선택합니다.
3. 규칙의 이름 및 설명을 입력합니다.

4. 이벤트 패턴을 선택한 다음 사용자 지정 패턴을 선택합니다. `"source": [ "aws.xray" ]` 및 `"detail-type": [ "AWS X-Ray Insight Update" ]`를 포함하는 패턴을 제공하십시오. 다음은 가능한 패턴의 몇 가지 예입니다.
   - X-Ray 인사이트에서 들어오는 모든 이벤트와 일치하는 이벤트 패턴:

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

   - 지정된 **`state`** 및 **`category`**와 일치하는 이벤트 패턴:

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

5. 이벤트가 이 규칙과 일치할 때 간접 호출할 대상을 선택하고 구성합니다.
6. (선택 사항) 이 규칙을 더 쉽게 식별하고 선택할 수 있도록 태그를 입력합니다.
7. 생성(Create)을 선택합니다.

> **Note**
>
> X-Ray 인사이트 알림은 현재 고객 관리형 키를 지원하지 않는 Amazon EventBridge로 이벤트를 전송합니다. 자세한 내용은 의 데이터 보호 AWS X-Ray 단원을 참조하십시오.

인사이트 개요

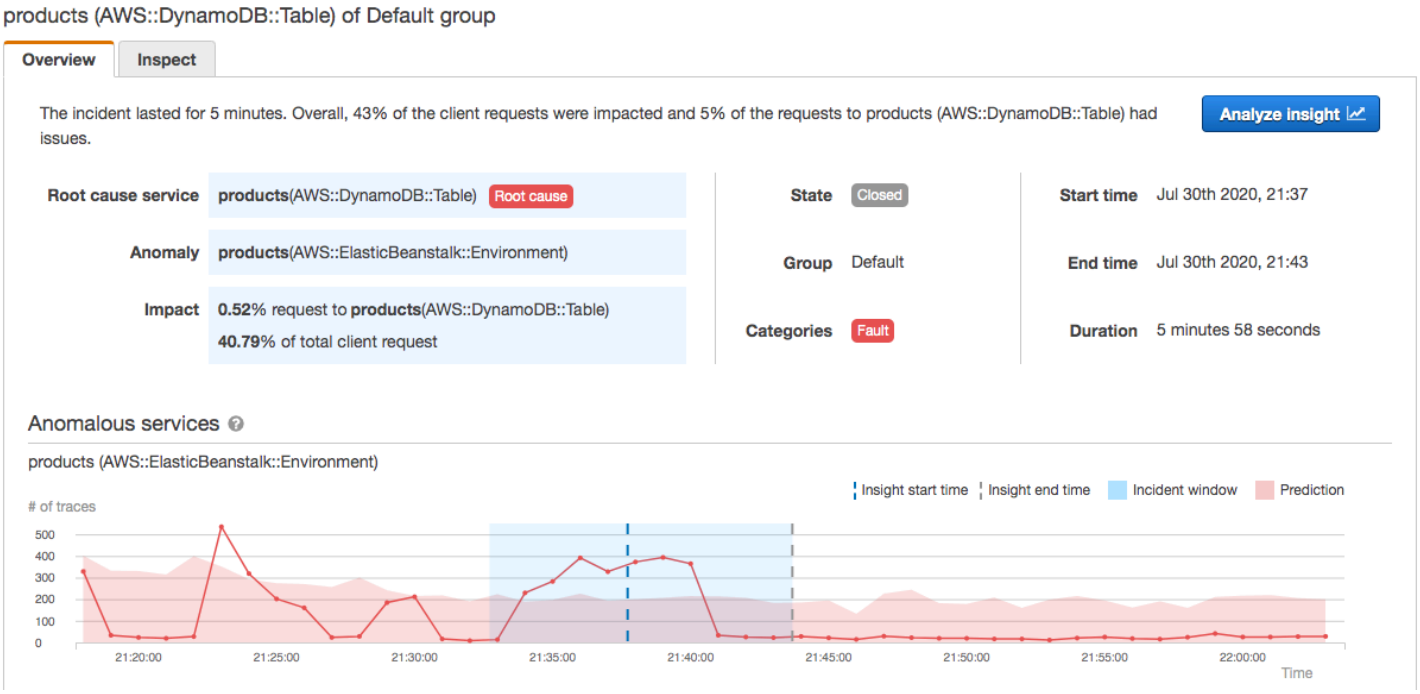
인사이트 개요 페이지에서는 세 가지 주요 질문에 대한 답변을 제시합니다:

- 근본적인 문제는 무엇인가요?
- 근본 원인은 무엇인가요?

- 어떤 영향이 있나요?

Anomalous Services 섹션에는 인시던트 중 결함률의 변화를 보여주는 각 서비스에 대한 타임라인이 표시됩니다. 타임라인에는 결함이 있는 추적의 개수와 기록된 트래픽 양에 따라 예상되는 결함 수를 나타내는 실선 밴드가 겹쳐져 표시됩니다. 인사이트 지속 시간은 인시던트 윈도우에 시각화되어 표시됩니다. 인시던트 윈도우는 X-Ray에서 지표가 비정상적으로 변하는 것을 관찰할 때 시작되며 인사이트가 활성화되어 있는 동안 지속됩니다.

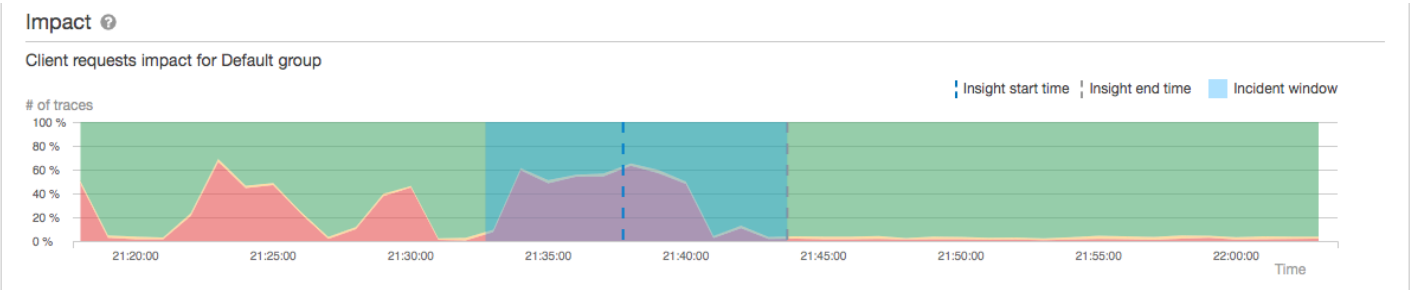
다음 예는 인시던트를 발생시킨 결함의 증가를 보여줍니다:



근본 원인 섹션에는 근본 원인 서비스와 영향을 받는 경로에 초점을 맞춘 트레이스 맵이 표시됩니다. 근본 원인 맵의 오른쪽 상단에 있는 눈 아이콘을 선택하면 영향을 받지 않은 노드를 숨길 수 있습니다. 근본 원인 서비스는 X-Ray가 이상 징후를 식별한 가장 멀리 위치한 다운스트림 노드입니다. 이는 사용자가 계측한 서비스 또는 서비스가 계측된 클라이언트로 호출한 외부 서비스를 나타낼 수 있습니다. 예를 들어 계측된 AWS SDK 클라이언트를 사용하여 Amazon DynamoDB를 호출하면 DynamoDB에서 장애가 증가하면 DynamoDB를 근본 원인으로 사용하여 인사이트를 얻을 수 있습니다.

근본 원인을 자세히 조사하려면 근본 원인 그래프에서 근본 원인 세부 정보 보기를 선택하십시오. 분석 페이지를 사용하여 근본 원인 및 관련 메시지를 조사할 수 있습니다. 자세한 내용은 분석 콘솔과 상호 작용 단원을 참조하십시오.

맵의 업스트림에서 계속해서 발생하는 오류는 여러 노드에 영향을 미치고 여러 이상 징후를 유발할 수 있습니다. 오류가 요청을 한 사용자에게까지 전달되면클라이언트 오류가 발생하게 됩니다. 트레이스 맵의 루트 노드에 결함이 있는 경우입니다. 영향 그래프는 전체 그룹에 대한 클라이언트 경험의 타임라인을 제공합니다. 이 경험은 장애, 오류, 스로틀, 정상 상태의 백분율을 기반으로 계산됩니다.



이 예는 인시던트 발생 시간 동안 루트 노드에서 결함이 발생하여 트레이스가 증가한 것을 보여줍니다. 다운스트림 서비스의 인시던트가 항상 클라이언트 오류의 증가와 일치하는 것은 아닙니다.

인사이트 분석을 선택하면 인사이트의 원인이 되는 일련의 트레이스에 대해 자세히 살펴볼 수 있는 X-Ray 분석 콘솔이 창에 열립니다. 자세한 내용은 분석 콘솔과 상호 작용 단원을 참조하십시오.

영향력 이해하기

AWS X-Ray 는 인사이트 및 알림 생성의 일환으로 진행 중인 문제로 인한 영향을 측정합니다. 영향력은 두 가지 방법으로 측정됩니다:

- X-Ray 그룹에 미치는 영향
- 근본 원인 서비스에 미치는 영향

이 영향력은 지정된 기간 내에 실패하거나 오류를 일으키는 요청의 비율에 따라 결정됩니다. 이 영향력 분석을 통해 특정 시나리오를 기반으로 문제의 심각도와 우선 순위를 도출할 수 있습니다. 이 영향력은 인사이트 알림 외에도 콘솔 환경의 일부로 제공됩니다.

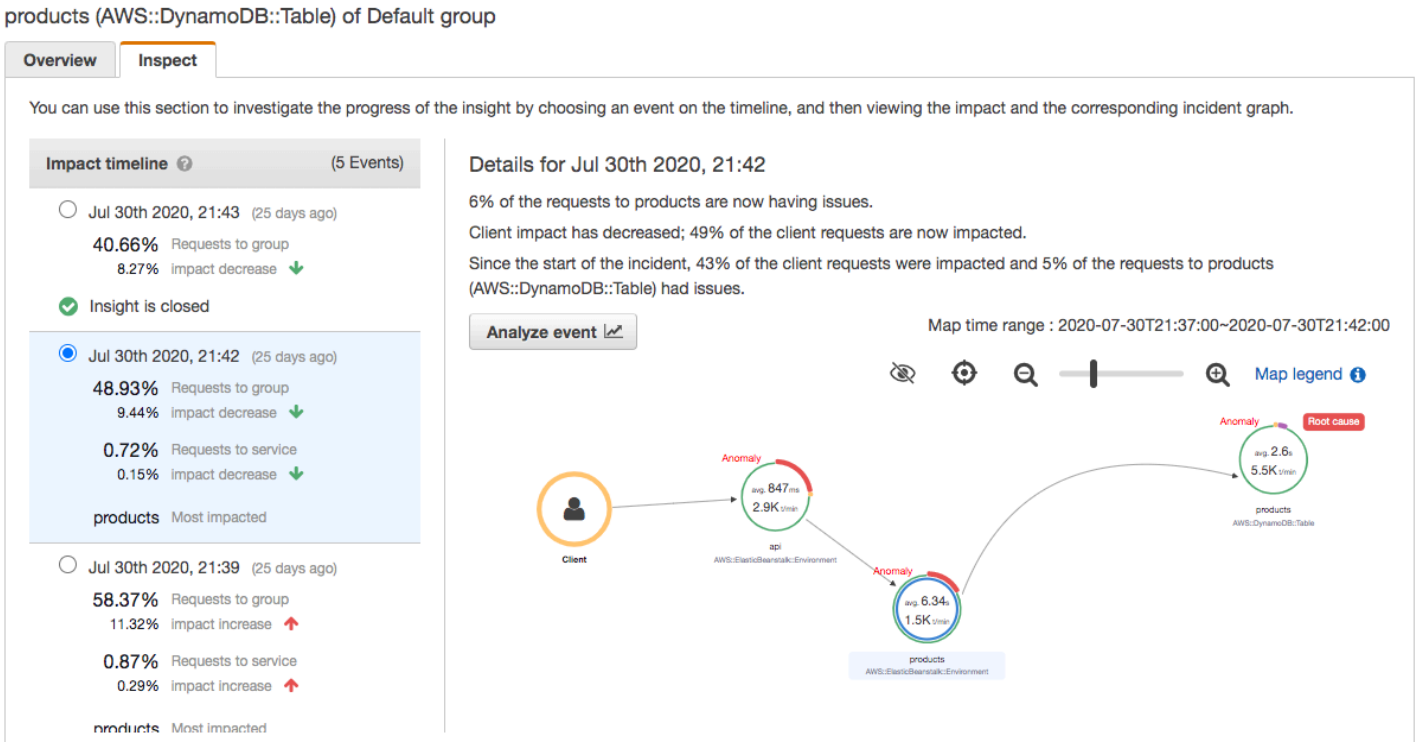
중복 제거

AWS X-Ray 인사이트는 여러 마이크로서비스에서 문제를 중복 제거합니다. 이상 징후 탐색을 사용하여 문제의 근본 원인이 되는 서비스를 파악하고, 동일한 근본 원인으로 인해 다른 관련 서비스에서 이상 행동을 보이는지 확인하고, 그 결과를 하나의 인사이트로 기록합니다.

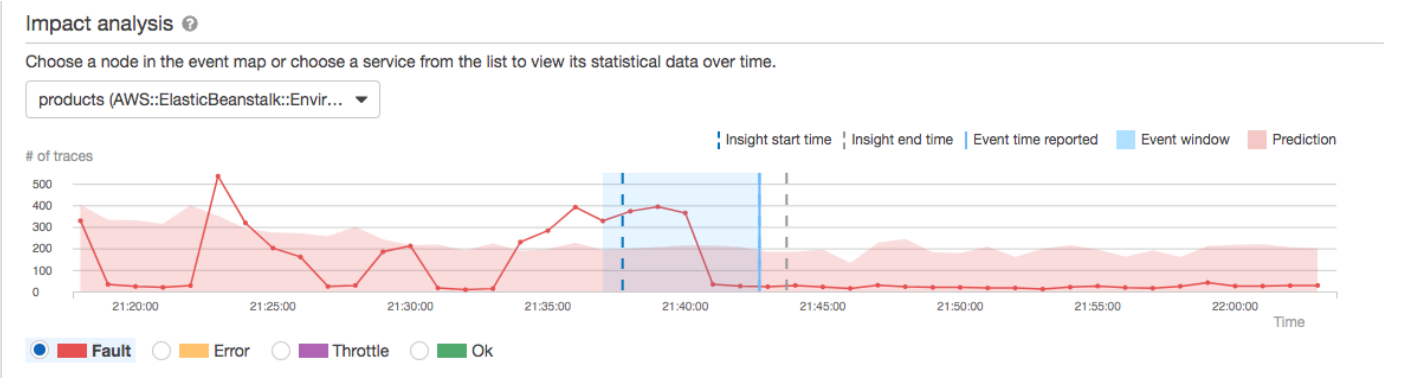
인사이트 진행 상황 검토

X-Ray는 인사이트가 해결될 때까지 주기적으로 재평가하고, 주목할 만한 중간 변경 사항을 알림으로 기록합니다. 알림은 Amazon EventBridge 이벤트로 전송할 수 있습니다. 이를 통해 프로세스와 워크플로를 구축하여 시간이 지남에 따라 이슈가 어떻게 변화했는지 파악하고, 이메일 전송 또는 EventBridge를 사용한 알림 시스템 통합 등의 적절한 조치를 취할 수 있습니다.

검사 페이지의 영향 타임라인에서 인시던트 이벤트를 검토할 수 있습니다. 다른 서비스를 선택할 때까지 타임라인에는 기본값으로 가장 영향을 많이 받는 서비스가 표시됩니다.



이벤트에 대한 트레이스 맵과 그래프를 보려면 영향 타임라인에서 해당 이벤트를 선택합니다. 트레이스 맵에는 인시던트의 영향을 받는 애플리케이션의 서비스가 표시됩니다. 영향 분석은 선택한 노드와 그룹 내 클라이언트의 장애 타임라인을 그래프로 표시합니다.



인시던트와 관련된 트레이스를 더 자세히 살펴보려면 검사 페이지에서 이벤트 분석을 선택하십시오. 분석 페이지를 사용하여 추적 목록을 세분화하고 영향을 받는 사용자를 식별할 수 있습니다. 자세한 내용은 분석 콘솔과 상호 작용 단원을 참조하십시오.

분석 콘솔과 상호 작용

AWS X-Ray 분석 콘솔은 애플리케이션과 기본 서비스의 성능을 빠르게 이해하기 위해 추적 데이터를 해석하는 대화형 도구입니다. 콘솔에서는 대화형 응답 시간 및 시계열 그래프를 통해 트레이스를 탐색, 분석 및 시각화할 수 있습니다.

분석 콘솔에서 선택하면 콘솔이 선택한 트레이스의 하위 집합까지 반영하여 필터를 생성합니다. 현재 트레이스 집합과 연결된 지표 및 필드의 그래프와 패널을 클릭하면 더욱 세부적인 필터로 활성 데이터 세트를 구체화할 수 있습니다.

주제

- 콘솔 기능
- 응답 시간 분포
- 시계열 활동
- 워크플로우 예시
- 서비스 그래프의 결함 관찰
- 응답 시간 피크 식별
- 상태 코드가 표시된 모든 추적 보기
- 사용자에게 연결된 하위 그룹 항목 모두 보기
- 서로 다른 기준을 사용한 추적 집합 2개의 비교
- 원하는 트레이스 식별 및 세부 정보 보기

콘솔 기능

X-Ray 분석 콘솔은 다음과 같이 트레이스 데이터의 그룹화, 필터링, 비교 및 정량화 같은 주요 기능을 사용합니다.

Features

Feature	설명
그룹	처음에 선택되는 그룹은 `Default`입니다. 검색된 그룹을 변경하려면 메인 필터 표현식 검색 창 오른쪽에 있는 메뉴에서 다른 그룹을 선택하십시오. 그룹에 대한 자세한 내용은 그룹에 필터 표현식 사용을 참조하십시오.
Retrieved traces(검색된 트레이스)	기본적으로 분석 콘솔은 선택된 그룹 내 모든 트레이스를 기준으로 그래프를 생성합니다. 검색된 트레이스란 유효 집합에 속한 모든 트레이스를 의미합니다. 트레이스 수는 현재 타일에서 찾을 수 있습니다. 메인 검색창에 적용하는 필터 표현식이 검색된 트레이스를 구체화하여 업데이트합니다.
Show in charts/Hide from charts(차트에 표시/차트에서 숨기기)	활성 그룹을 검색된 트레이스와 비교할 수 있는 토글입니다. 그룹 관련 데이터를 활성 필터와 비교하려면 Show in charts(차트에 표시)를 선택합니다. 현재 뷰를 차트에서 제거하려면 Hide from charts(차트에서 숨기기)를 선택합니다.
Filtered trace set A(필터링된 트레이스 집합 A)	그래프 및 표와의 상호 작용을 통해 필터를 적용하여 필터링된 추적 세트 A의 기준을 생성합니다. 필터가 적용되면 이 타일 내에서 적용 가능한 추적의 수와 검색된 전체에서 차지하는 추적의 비율이 계산됩니다. 필터가 Filtered trace set A(필터링된 추적 집합 A) 타일 내에서 태그로 채워지며, 타일에서 필터를 제거할 수도 있습니다.
Refine(구체화)	이 기능은 검색된 추적 집합을 추적 집합 A에 적용되는 필터를 기준으로 업데이트합니다. 검색

Feature	설명
	된 추적 집합을 구체화하면 추적 집합 A의 필터를 기준으로 검색된 모든 추적의 유효 집합이 새로 고쳐집니다. 검색된 추적의 유효 집합이란 그룹에 속한 모든 추적의 하위 집합 샘플을 말합니다.
Filtered trace set B(필터링된 트레이스 집합 B)	필터링된 추적 세트 B가 생성되면 필터링된 추적 세트 A의 복사본이 됩니다. 두 추적 세트를 비교하려면 추적 세트 A는 고정된 상태로 유지하면서 추적 세트 B에 적용할 새 필터를 선택합니다. 필터가 적용되면 사용 가능한 트레이스 수와 검색된 트레이스의 전체 대비 비율이 현재 타일 내에서 계산됩니다. 필터가 Filtered trace set B(필터링된 추적 집합 B) 타일 내에서 태그로 채워지며, 타일에서 필터를 제거할 수도 있습니다.
Response Time Root Cause Entity Paths(응답 시간 근본 원인 개체 경로)	기록된 개체 경로의 테이블입니다. X-Ray는 트레이스에서 응답 시간 원인으로 가장 가능성이 높은 경로가 무엇인지 결정합니다. 포맷은 발생하는 개체 계층 구조를 나타내며, 응답 시간 근본 원인으로 끝납니다. 이러한 계층 구조의 행을 사용해 반복되는 응답 시간 결함을 필터링합니다. 근본 원인 필터를 사용자 지정하고, API를 통해 데이터를 가져오는 방법에 대한 자세한 내용은 근본 원인 분석 가져오기 및 구체화 단원을 참조하십시오.
델타(�)	추적 집합 A와 추적 집합 B가 모두 활성 상태일 때 지표 테이블에 추가되는 열입니다. 델타 열은 추적 집합 A와 추적 집합 B의 추적 비율 차이를 계산합니다.

응답 시간 분포

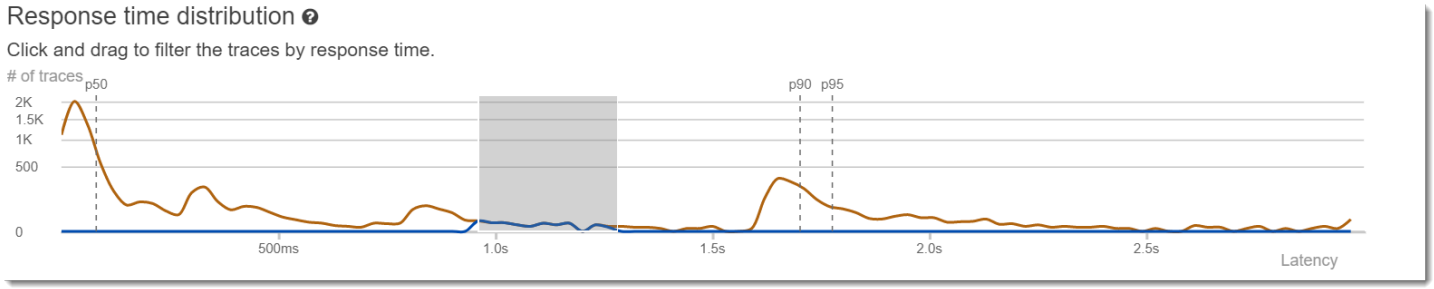
분석 콘솔은 기본적으로 두 가지 그래프인 Response Time Distribution(응답 시간 분포)와 Time Series Activity(시계열 활동)을 생성하여 트레이스를 시각화합니다. 이번 섹션과 다음 섹션에서는 각 그래프의 예를 살펴보고 그래프를 읽는 기본적인 방법에 대해서 살펴보겠습니다.

다음은 응답 시간 선 그래프와 연결되는 색상을 나타낸 것입니다(시계열 그래프는 동일한 색상 체계를 사용함).

- All traces in the group(그룹 내 모든 트레이스 – 회색
- Retrieved traces(검색된 트레이스) – 주황색
- Filtered trace set A(필터링된 트레이스 집합 A) – 녹색
- Filtered trace set B(필터링된 트레이스 집합 B) – 파란색

Example – 응답 시간 분포

응답 시간 분포란 임의 응답 시간을 갖는 트레이스 수를 나타낸 차트를 말합니다. 응답 시간 분포 내에서 선택하려면 클릭하여 끌어오면 됩니다. 그러면 특정 응답 시간에 속하는 모든 트레이스를 대상으로 `responseTime`이라고 하는 유효 트레이스 집합에 필터가 선택 및 생성됩니다.

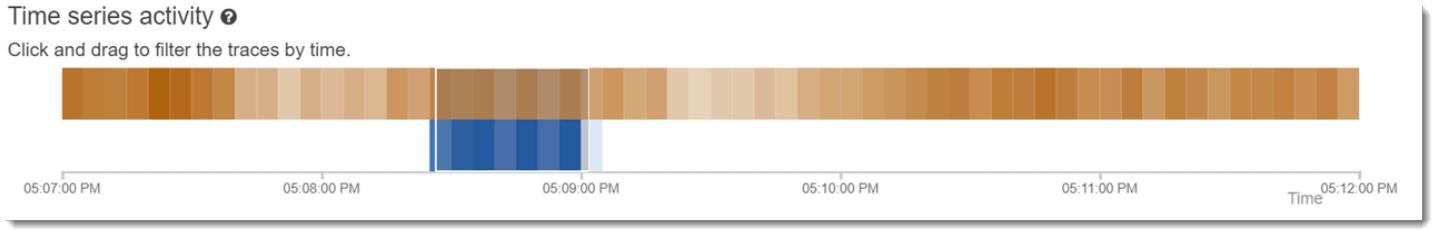


시계열 활동

시계열 활동 차트에는 특정 시간의 범위에 속하는 트레이스 수가 표시됩니다. 색상 표시기에는 응답 시간 분포의 선 그래프 색상이 그대로 반영됩니다. 연속된 활동 내에서 색상 블록이 더욱 어둡고 진할수록 해당 시간에 트레이스가 더욱 많다는 것을 의미합니다.

Example – 시계열 활동

시계열 활동 그래프 내에서 선택하려면 클릭하고 드래그하면 됩니다. 그러면 특정 시간 범위에 속하는 모든 트레이스를 대상으로 유효 트레이스 집합에 `timerange`이라고 하는 필터가 선택 및 생성됩니다.



워크플로우 예시

다음은 X-Ray 분석 콘솔의 공통 사용 사례를 나타낸 예시입니다. 각 예시는 콘솔 환경의 주요 기능을 설명하고 있습니다. 또한 그룹으로서 기본적인 문제 해결 워크플로우를 따릅니다. 이 단계에서는 비정상 노드를 먼저 찾아낸 다음 Analytics 콘솔과 상호 작용하여 비교 쿼리를 자동으로 생성하는 방법을 안내합니다. 쿼리를 통해 범위를 좁혔으면 마지막으로 관심 있는 추적을 자세히 살펴보고 서비스 상태를 저하시키는 것이 무엇인지 파악합니다.

서비스 그래프의 결함 관찰

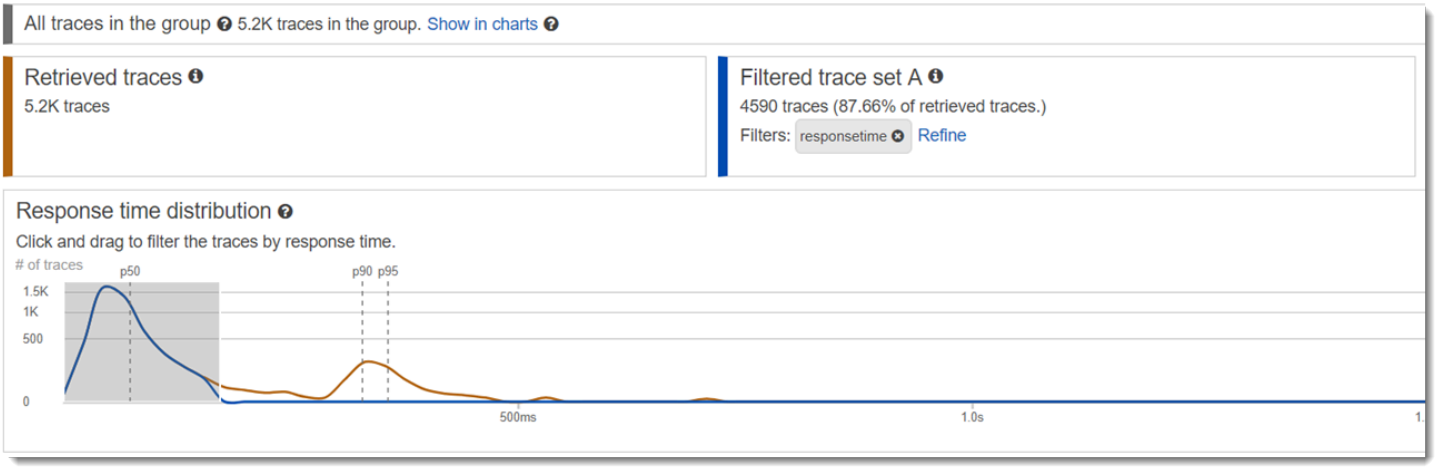
이 트레이스 맵은 오류 및 결함 직접 호출 성공 비율에 따라 각 노드의 색상을 다르게 표시해 노드의 상태를 표시합니다. 노드에서 빨간색 비율은 결함을 의미합니다. 이때는 X-Ray 분석 콘솔을 사용해 조사하십시오.

트레이스 맵을 읽는 방법에 대한 자세한 내용은 트레이스 맵 보기를 참조하세요.

응답 시간 피크 식별

응답 시간 분포를 사용해 응답 시간 피크를 관찰할 수 있습니다. 응답 시간 피크를 선택하면 그래프 아래 테이블이 업데이트되면서 상태 코드 등 연결된 지표가 모두 표시됩니다.

클릭하고 끌면 X-Ray에서 필터를 선택하고 생성합니다. 그래프의 선 위에 회색 그림자로 표시됩니다. 이제 해당 그림자를 분포 그래프를 따라 왼쪽 또는 오른쪽으로 끌어서 선택 항목과 필터를 업데이트할 수 있습니다.



상태 코드가 표시된 모든 추적 보기

그래프 아래 지표 테이블을 사용해 선택한 피크 범위에 속하는 트레이스를 자세하게 살펴볼 수 있습니다. HTTP STATUS CODE 테이블에서 행을 클릭하면 유효한 데이터 세트에 필터가 자동으로 생성됩니다. 예를 들어 상태 코드가 500인 트레이스를 모두 볼 수 있습니다. 동시에 트레이스 집합 타일에 `http.status`라는 이름으로 필터 태그가 생성됩니다.

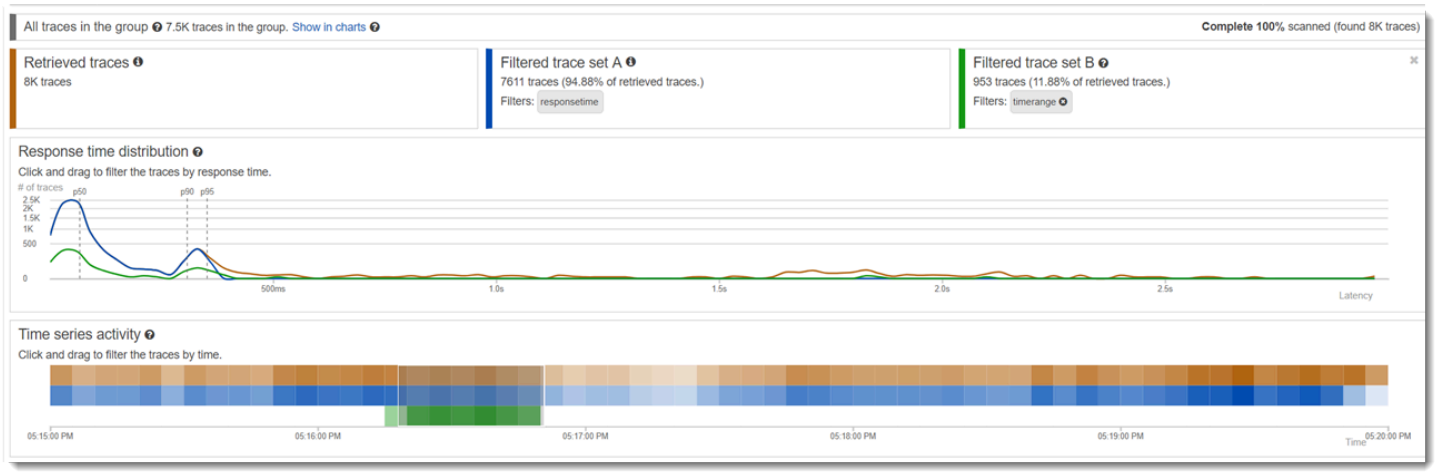
사용자에게 연결된 하위 그룹 항목 모두 보기

사용자, URL, 응답 시간 근본 원인 또는 기타 사전 설정된 속성을 기준으로 오류 집합을 자세하게 살펴봅니다. 예를 들어 상태 코드가 500인 트레이스 집합을 추가로 필터링한다고 가정할 경우, 먼저 USERS 테이블에서 행을 선택합니다. 그러면 트레이스 집합 타일에 앞서 지정했던 `http.status`와 `user`까지 필터 태그 2개가 생성됩니다.

서로 다른 기준을 사용한 추적 집합 2개의 비교

다양한 사용자와 사용자의 POST 요청을 비교하여 불일치 및 상관관계를 찾습니다. 첫 번째 필터 집합을 적용합니다. 이 집합은 응답 시간 분포 그래프에서 파란색 선으로 정의됩니다. 그런 다음 비교를 선택합니다. 그러면 처음에는 트레이스 집합 A에 필터 복사본이 생성됩니다.

계속 하려면 트레이스 집합 B에 적용할 필터 집합을 새로 정의합니다. 두 번째 집합은 녹색 선으로 표시됩니다. 아래 예에서는 파란색 및 녹색 체계에 따라 다른 선으로 표시되어 있습니다.



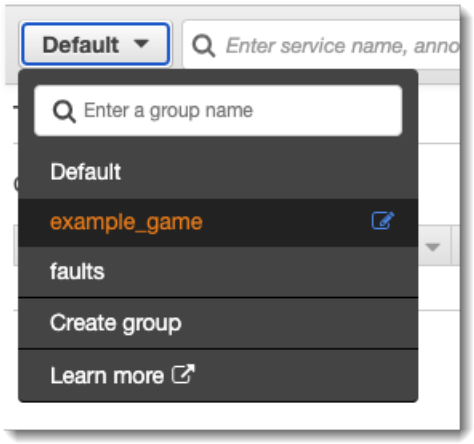
원하는 트레이스 식별 및 세부 정보 보기

콘솔 필터를 사용해 범위를 좁히면 지표 테이블 아래 트레이스 목록이 더 의미있게 바뀝니다. 추적 목록 테이블은 URL, USER 및 STATUS CODE에 대한 정보를 결합하여 단일 뷰로 표시합니다. 더 많은 세부 정보를 원한다면 테이블에서 행을 선택하여 트레이스 세부 정보 페이지를 열고 타임라인과 원시 데이터를 확인합니다.

그룹 구성

그룹은 필터 표현식으로 정의한 추적 모음입니다. 그룹을 사용하여 추가 서비스 그래프를 작성하여 Amazon CloudWatch 지표를 공급할 수 있습니다. AWS X-Ray 콘솔 또는 X-Ray API를 사용하여 서비스에 대한 그룹을 만들고 관리할 수 있습니다. 이 항목에서는 X-Ray 콘솔을 사용하여 그룹을 만들고 관리하는 방법에 대해 설명합니다. X-Ray API를 사용하여 그룹을 관리하는 방법에 대한 자세한 내용은 Groups을 참조하십시오.

트레이스 맵, 트레이스 또는 분석을 위한 트레이스 그룹을 생성할 수 있습니다. 그룹을 생성하면 트레이스 맵, 트레이스, 분석의 세 페이지 모두에 있는 그룹 드롭다운 메뉴에서 해당 그룹을 필터로 사용할 수 있습니다.



그룹은 이름 또는 Amazon 리소스 이름(ARN)으로 식별되며 필터 표현식을 포함합니다. 이 서비스는 수신 트레이스를 표현식과 비교하여 그에 따라 저장합니다. 필터 표현식을 작성하는 방법에 대한 자세한 내용은 필터 표현식 사용을 참조하십시오.

그룹의 필터 표현식을 업데이트해도 이미 기록한 데이터는 변경되지 않습니다. 업데이트는 후속 추적에만 적용됩니다. 이렇게 하면 새 표현식과 이전 표현식의 그래프를 병합할 수 있습니다. 이를 방지하려면 현재 그룹을 삭제하고 새로 만드십시오.

> **Note**
>
> 그룹은 필터 표현식과 일치하는 검색 완료 트레이스의 수로 청구됩니다. 자세한 내용은 AWS X-Ray 요금을 참조하세요.

주제

- 그룹 생성
- 그룹 적용
- 그룹 편집
- 그룹 복제
- 그룹 삭제
- Amazon CloudWatch에서 그룹 지표 보기

그룹 생성

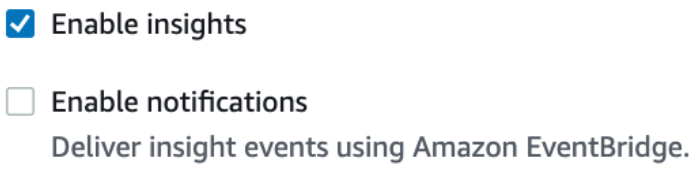
> **Note**
>
> 이제 Amazon CloudWatch 콘솔 내에서 X-Ray 그룹을 구성할 수 있습니다. X-Ray 콘솔을 계속 사용할 수도 있습니다.

CloudWatch console

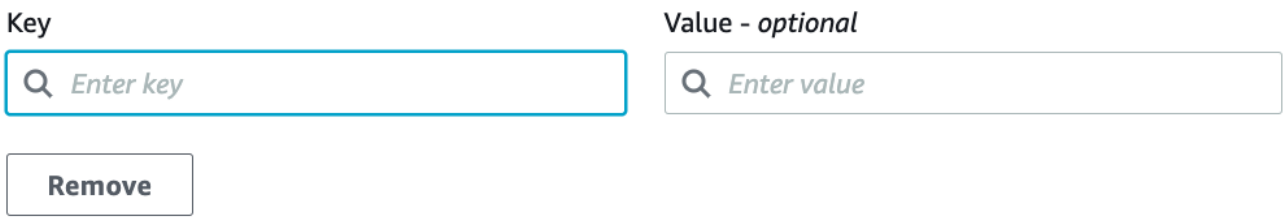
1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/cloudwatch/:// https://https://://https://://https://://CloudWatch://://https://://https://://https://https://://https://://https
2. 왼쪽 탐색 창에서 설정을 선택합니다.
3. X-Ray 추적 섹션의 그룹에서 설정 보기를 선택합니다.
4. 그룹 목록 위에 있는 그룹 만들기를 선택합니다.
5. Create group(그룹 생성) 페이지에서 그룹의 이름을 입력합니다. 그룹 이름은 최대 32자여야 하며 영숫자와 대시를 포함할 수 있습니다. 그룹 이름은 대/소문자를 구분합니다.
6. 필터 표현식을 입력하세요. 필터 표현식을 작성하는 방법에 대한 자세한 내용은 필터 표현식 사용을 참조하십시오. 다음 예제에서 그룹은 서비스의 `api.example.com`에서 오류 추적과 응답 시간이 5초 이상인 서비스에 대한 요청을 필터링합니다.

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

7. Insights(인사이트)에서 그룹에 대한 인사이트 액세스를 활성화하거나 비활성화합니다. Insights(인사이트)에 대한 자세한 내용은 X-Ray 인사이트 사용 단원을 참조하십시오.

8. 태그에서 새 태그 추가를 선택하여 태그 키를 입력하고 선택적으로 태그 값을 입력합니다. 원하는 대로 태그를 계속 추가합니다. 각 태그 키는 고유해야 합니다. 태그를 삭제하려면 Remove(제거)를 선택합니다. 태그에 대한 자세한 내용은 X-Ray 샘플링 규칙 및 그룹 태그 지정하기 섹션을 참조하세요.



9. 그룹 생성을 선택합니다.

X-Ray console

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/xray/home:// https://https://https://https://://https://://https://://://https://www.
2. 왼쪽 탐색 창의 그룹 페이지 또는 트레이스 맵, 트레이스, 분석 페이지 중 하나의 그룹 메뉴에서 그룹 생성 페이지를 엽니다.
3. Create group(그룹 생성) 페이지에서 그룹의 이름을 입력합니다. 그룹 이름은 최대 32자여야 하며 영숫자와 대시를 포함할 수 있습니다. 그룹 이름은 대/소문자를 구분합니다.
4. 필터 표현식을 입력하세요. 필터 표현식을 작성하는 방법에 대한 자세한 내용은 필터 표현식 사용을 참조하십시오. 다음 예제에서 그룹은 서비스의 `api.example.com`에서 오류 추적과 응답 시간이 5초 이상인 서비스에 대한 요청을 필터링합니다.

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

5. Insights(인사이트)에서 그룹에 대한 인사이트 액세스를 활성화하거나 비활성화합니다. Insights(인사이트)에 대한 자세한 내용은 X-Ray 인사이트 사용 단원을 참조하십시오.

6. 태그에서 태그 키와 태그 값(선택 사항)을 입력합니다. 태그를 추가하면 다른 태그를 입력할 수 있는 새 줄이 나타납니다. 각 태그 키는 고유해야 합니다. 태그를 삭제하려면 태그 행 끝에 있는 X를 선택합니다. 태그에 대한 자세한 내용은 X-Ray 샘플링 규칙 및 그룹 태그 지정하기 섹션을 참조하세요.

7. 그룹 생성을 선택합니다.

그룹 적용

CloudWatch console

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/cloudwatch/:// https://https://https://://https://://https://://https://://https://://CloudWatch://https://https://://https:// http
2. 탐색 창의 X-Ray traces에서 다음 페이지 중 하나를 엽니다.
   - 트레이스 맵
   - 트레이스
3. X-Ray 그룹으로 필터링 필터에 그룹 이름을 입력합니다. 페이지에 표시되는 데이터는 그룹에 설정된 필터 표현식과 일치하도록 변경됩니다.

X-Ray console

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/xray/home:// https://https://https://https://://https://://https://://://https://www.
2. 탐색 창에서 다음 페이지 중 하나를 엽니다:
   - 트레이스 맵
   - 트레이스
   - 분석
3. 그룹 메뉴에서 the section called “그룹 생성”에서 생성한 그룹을 선택합니다. 페이지에 표시되는 데이터는 그룹에 설정된 필터 표현식과 일치하도록 변경됩니다.

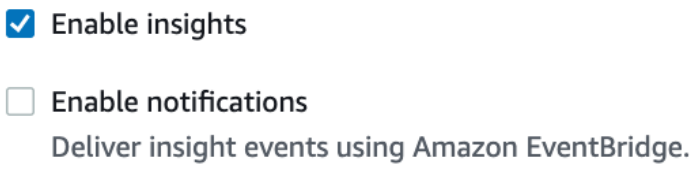
그룹 편집

CloudWatch console

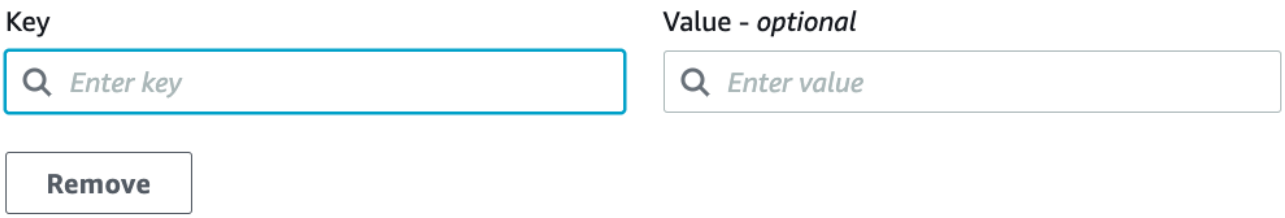
1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/cloudwatch/:// https://https://://https://://https://://CloudWatch://://https://://https://://https://https://://https://:// https
2. 왼쪽 탐색 창에서 설정을 선택합니다.
3. X-Ray 추적 섹션의 그룹에서 설정 보기를 선택합니다.
4. 그룹 섹션에서 그룹을 선택한 다음 편집을 선택합니다.
5. 그룹 이름을 바꿀 수는 없지만 필터 표현식은 업데이트할 수 있습니다. 필터 표현식을 작성하는 방법에 대한 자세한 내용은 필터 표현식 사용을 참조하십시오. 다음 예제에서 그룹은 요청 URL 주소에 `example/game`가 포함되어 있고 요청에 대한 응답 시간이 5초 이상인 `api.example.com` 서비스에서 오류 추적을 필터링합니다.

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

6. Insights(인사이트)에서 그룹에 대한 인사이트 액세스를 활성화하거나 비활성화합니다. Insights(인사이트)에 대한 자세한 내용은 X-Ray 인사이트 사용 단원을 참조하십시오.

7. 태그에서 새 태그 추가를 선택하여 태그 키를 입력하고 선택적으로 태그 값을 입력합니다. 원하는 대로 태그를 계속 추가합니다. 각 태그 키는 고유해야 합니다. 태그를 삭제하려면 Remove(제거)를 선택합니다. 태그에 대한 자세한 내용은 X-Ray 샘플링 규칙 및 그룹 태그 지정하기 섹션을 참조하세요.

8. 그룹 업데이트를 마치면 그룹 업데이트를 선택합니다.

X-Ray console

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/xray/home:// https://https://https://://https://://https://://://https://://://www.
2. 다음 중 하나를 수행하여 그룹 편집 페이지를 엽니다.

   a. 그룹 페이지에서 편집할 그룹 이름을 선택합니다.

   b. 다음 페이지 중 하나의 그룹 메뉴에서 그룹을 가리킨 다음 편집을 선택합니다.

   - 트레이스 맵
   - 트레이스
   - 분석

3. 그룹 이름을 바꿀 수는 없지만 필터 표현식은 업데이트할 수 있습니다. 필터 표현식을 작성하는 방법에 대한 자세한 내용은 필터 표현식 사용을 참조하십시오. 다음 예제에서 그룹은 요청 URL 주소에 `example/game`가 포함되어 있고 요청에 대한 응답 시간이 5초 이상인 `api.example.com` 서비스에서 오류 추적을 필터링합니다.

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

4. Insights(인사이트)에서 그룹에 대한 통찰력 및 통찰력 알림을 활성화하거나 비활성화합니다. Insights(인사이트)에 대한 자세한 내용은 X-Ray 인사이트 사용 단원을 참조하십시오.

5. 태그에서 태그 키 및 값을 편집합니다. 각 태그 키는 고유해야 합니다. 태그 값은 선택 사항이며 원하는 경우 값을 삭제할 수 있습니다. 태그를 삭제하려면 태그 행 끝에 있는 X를 선택합니다. 태그에 대한 자세한 내용은 X-Ray 샘플링 규칙 및 그룹 태그 지정하기 섹션을 참조하세요.

6. 그룹 업데이트를 마치면 그룹 업데이트를 선택합니다.

그룹 복제

그룹을 복제하면 기존 그룹의 필터 표현식과 태그가 포함된 새 그룹이 생성됩니다. 그룹을 복제하면 새 그룹은 해당 그룹이 복제된 그룹과 이름이 같으며 이름 뒤에 `-clone`이 추가됩니다.

CloudWatch console

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/cloudwatch/:// https://https://://https://://https://://CloudWatch://://https://://https://://https://https://://https://:// https
2. 왼쪽 탐색 창에서 설정을 선택합니다.
3. X-Ray 추적 섹션의 그룹에서 설정 보기를 선택합니다.
4. 그룹 섹션에서 그룹을 선택한 다음 복제을 선택합니다.
5. 그룹 생성 페이지에서 그룹 이름은 `####-clone`입니다. 그룹의 새 이름을 입력합니다(선택 사항). 그룹 이름은 최대 32자여야 하며 영숫자와 대시를 포함할 수 있습니다. 그룹 이름은 대/소문자를 구분합니다.
6. 기존 그룹의 필터 표현식을 그대로 유지하거나 선택적으로 새 필터 표현식을 입력할 수 있습니다. 필터 표현식을 작성하는 방법에 대한 자세한 내용은 필터 표현식 사용을 참조하십시오. 다음 예제에서 그룹은 서비스의 `api.example.com`에서 오류 추적과 응답 시간이 5초 이상인 서비스에 대한 요청을 필터링합니다.

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

7. 필요한 경우 태그에서 태그 키 및 값을 편집합니다. 각 태그 키는 고유해야 합니다. 태그 값은 선택 사항이며 원하는 경우 값을 삭제할 수 있습니다. 태그를 삭제하려면 태그 행 끝에 있는 X를 선택합니다. 태그에 대한 자세한 내용은 X-Ray 샘플링 규칙 및 그룹 태그 지정하기 섹션을 참조하세요.
8. 그룹 생성을 선택합니다.

X-Ray console

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/xray/home:// https://https://https://://https://://https://://://https://://://www.
2. 왼쪽 탐색 창에서 그룹 페이지를 열고 복제할 그룹 이름을 선택합니다.
3. [작업(Actions)] 메뉴에서 [그룹 복제(Clone group)]를 선택합니다.

4. 그룹 생성 페이지에서 그룹 이름은 ####-clone입니다. 그룹의 새 이름을 입력합니다(선택 사항). 그룹 이름은 최대 32자여야 하며 영숫자와 대시를 포함할 수 있습니다. 그룹 이름은 대/소문자를 구분합니다.
5. 기존 그룹의 필터 표현식을 그대로 유지하거나 선택적으로 새 필터 표현식을 입력할 수 있습니다. 필터 표현식을 작성하는 방법에 대한 자세한 내용은 필터 표현식 사용을 참조하십시오. 다음 예제에서 그룹은 서비스의 api.example.com에서 오류 추적과 응답 시간이 5초 이상인 서비스에 대한 요청을 필터링합니다.

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

6. 필요한 경우 태그에서 태그 키 및 값을 편집합니다. 각 태그 키는 고유해야 합니다. 태그 값은 선택 사항이며 원하는 경우 값을 삭제할 수 있습니다. 태그를 삭제하려면 태그 행 끝에 있는 X를 선택합니다. 태그에 대한 자세한 내용은 X-Ray 샘플링 규칙 및 그룹 태그 지정하기 섹션을 참조하세요.
7. 그룹 생성을 선택합니다.

그룹 삭제

이 섹션의 단계를 따르면 그룹을 삭제할 수 있습니다. 기본 그룹은 삭제할 수 없습니다.

CloudWatch console

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/cloudwatch/:// https://https://://https://://https://://CloudWatch://://https://://https://://https://https://://https://:// https
2. 왼쪽 탐색 창에서 설정을 선택합니다.
3. X-Ray 추적 섹션의 그룹에서 설정 보기를 선택합니다.
4. 그룹 섹션에서 그룹을 선택한 다음 삭제를 선택합니다.
5. 확인 메시지가 표시되면 삭제를 선택합니다.

X-Ray console

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/xray/home:// https://https://https://https://://https://://https://://://https://://www.
2. 왼쪽 탐색 창에서 그룹 페이지를 열고 삭제할 그룹 이름을 선택합니다.
3. 작업 메뉴에서 그룹 삭제를 선택합니다.

4. 확인 메시지가 표시되면 삭제를 선택합니다.

Amazon CloudWatch에서 그룹 지표 보기

그룹이 생성된 후 수신 추적은 X-Ray 서비스에 저장될 때 그룹의 필터 표현식에 대해 확인됩니다. 각 기준과 일치하는 추적 수에 대한 지표는 매분마다 Amazon CloudWatch에 게시됩니다. 그룹 편집 페이지에서 지표 보기를 선택하면 CloudWatch 콘솔에서 지표 페이지가 열립니다. CloudWatch 지표를 사용하는 방법에 대한 자세한 내용은 Amazon CloudWatch 사용 설명서의 Amazon CloudWatch 지표 사용을 참조하세요.

CloudWatch console

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/cloudwatch/:// https://https://https://://https://://https://://https://://https://://https://https://:// CloudWatch ://
2. 왼쪽 탐색 창에서 설정을 선택합니다.
3. X-Ray 추적 섹션의 그룹에서 설정 보기를 선택합니다.
4. 그룹 섹션에서 그룹을 선택한 다음 편집을 선택합니다.
5. 그룹 편집 페이지에서 지표 보기을 선택합니다.

   그러면 Amazon CloudWatch 콘솔에 지표 페이지가 열립니다.

X-Ray console

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/xray/home:// https://https://https://://https://://https://://://https://://://www.
2. 왼쪽 탐색 창에서 그룹 페이지를 열고 지표를 보려는 그룹의 이름을 선택합니다.
3. 그룹 편집 페이지에서 지표 보기을 선택합니다.

   그러면 Amazon CloudWatch 콘솔에 지표 페이지가 열립니다.

샘플링 규칙 구성

AWS X-Ray 콘솔을 사용하여 서비스에 대한 샘플링 규칙을 구성할 수 있습니다. 샘플링 구성으로 활성 추적 AWS 서비스 을 지원하는 X-Ray SDK 및는 샘플링 규칙을 사용하여 기록할 요청을 결정합니다.

주제

샘플링 규칙 구성

다음 사용 사례에 대한 샘플링을 구성할 수 있습니다.

- API 게이트웨이 진입점 — API 게이트웨이는 샘플링 및 활성 추적을 지원합니다. API 단계에서 적극 추적을 활성화하려면 에 대한 Amazon API Gateway 활성 추적 지원 AWS X-Ray 단원을 참조하십시오.
- AWS AppSync - 샘플링 및 활성 추적을 AWS AppSync 지원합니다. 요청에 대해 AWS AppSync 활성 추적을 활성화하려면 AWS X-Ray로 추적을 참조하세요.
- 컴퓨팅 플랫폼에서 X-Ray SDK 계측 - Amazon EC2, Amazon ECS 또는 같은 컴퓨팅 플랫폼을 사용하는 경우 애플리케이션이 최신 X-Ray SDK로 계측되면 AWS Elastic Beanstalk샘플링이 지원됩니다.

샘플링 규칙 사용자 정의

샘플링 규칙을 사용자 지정하여 기록하는 데이터의 양을 제어할 수 있습니다. 코드를 수정하거나 다시 배포하지 않고도 샘플링 동작을 수정할 수도 있습니다. 샘플링 규칙은 X-Ray SDK에 기준 세트에 대해 얼마나 많은 요청을 기록할지 알려줍니다. 기본적으로 X-Ray SDK는 매초 첫 번째 요청과 추가 요청의 5%를 기록합니다. 초당 하나의 요청은 리저버입니다. 이는 서비스가 요청을 처리 중인 동안 하나 이상의 트레이스가 매초 기록되도록 합니다. 5퍼센트는 리저버 크기를 넘는 추가 요청이 샘플링되는 비율입니다.

코드에 포함시키는 JSON 문서에서 샘플링 규칙을 읽도록 X-Ray SDK를 구성할 수 있습니다. 그러나 여러 서비스 인스턴스를 실행할 때 각 인스턴스는 독립적으로 샘플링을 수행합니다. 이로 인해 샘플링되는 요청의 전체 백분율이 증가합니다. 모든 인스턴스의 리저버가 함께 효과적으로 추가되기 때문입니다. 또한 로컬 샘플링 규칙을 업데이트하려면 코드를 다시 배포해야 합니다.

X-Ray 콘솔에서 샘플링 규칙을 정의하고 X-Ray 서비스에서 규칙을 읽도록 SDK를 구성하면 이러한 문제를 모두 피할 수 있습니다. 서비스는 각 규칙의 리저버를 관리하고, 서비스의 각 인스턴스에 할당량을 할당하여 실행 중인 인스턴스의 수에 따라 리저버를 고르게 배포합니다. 리저버 제한은 설정한 규칙에 따라 계산됩니다. 그리고 규칙이 서비스에 구성되어 있기 때문에 추가로 배포하지 않고도 규칙을 관리할 수 있습니다.

> **Note**
>
> X-Ray는 샘플링 규칙을 적용할 때 최선의 방법을 사용하며, 경우에 따라 유효 샘플링 속도가 구성된 샘플링 규칙과 정확히 일치하지 않을 수 있습니다. 하지만 시간이 지나면서 샘플링된 요청 수는 구성된 비율에 가까워야 합니다.

이제 Amazon CloudWatch 콘솔 내에서 X-Ray 샘플링 규칙을 구성할 수 있습니다. X-Ray 콘솔을 계속 사용할 수도 있습니다.

CloudWatch console

CloudWatch 콘솔에서 샘플링 규칙을 구성하려면

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/cloudwatch/:// https://https://https://://https://https://https://://https://https://://https://https://https://:// CloudWatch ://
2. 왼쪽 탐색 창에서 설정을 선택합니다.
3. X-Ray 추적 섹션의 샘플링 규칙에서 설정 보기를 선택합니다.
4. 샘플링 규칙 생성을 선택하여 규칙을 생성합니다.

   규칙을 편집하려면 규칙을 선택한 다음 편집을 선택하여 편집합니다.

   규칙을 삭제하려면 규칙을 선택하고 삭제 를 선택하여 삭제합니다.

X-Ray console

X-Ray 콘솔에서 샘플링 규칙을 구성하려면

1. X-Ray 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 Sampling(샘플링)을 선택합니다.
3. 샘플링 규칙 생성을 선택하여 규칙을 생성합니다.

규칙 이름을 선택하여 규칙을 편집합니다.

규칙을 선택하고 작업메뉴를 사용하여 규칙을 삭제합니다.

샘플링 규칙 옵션

다음과 같은 옵션을 각 규칙에 사용할 수 있습니다. 문자열 값은 와일드카드를 사용하여 단일 문자(?) 또는 0개 이상의 문자(*)와 일치시킬 수 있습니다.

샘플링 규칙 옵션

- 규칙 이름(문자열) - 규칙의 고유한 이름입니다.
- 우선 순위(1~9999 사이의 정수) - 샘플링 규칙의 우선 순위입니다. 서비스에서 규칙의 우선 순위를 오름차순으로 평가하며 일치하는 첫 번째 규칙으로 샘플링을 결정합니다.
- 리저버 (음수가 아닌 정수) – 고정 비율을 적용하기 전의 초당 구성과 일치하는 요청의 일정한 수. 리저버는 서비스에서 직접 사용하지 않지만 규칙을 총체적으로 사용하여 모든 서비스에 적용됩니다.
- 속도(0~100 사이의 정수) - 리저버가 소진된 후, 계측과 일치하는 요청의 백분율입니다. 콘솔에서 샘플링 규칙을 구성할 때 0에서 100 사이의 백분율을 선택합니다. JSON 문서를 사용하여 클라이언트 SDK에서 샘플링 규칙을 구성할 때는 0에서 1 사이의 백분율 값을 입력합니다.
- 서비스 이름(문자열) – 트레이스 맵에 표시되는 계측된 서비스의 이름입니다.
  - X-Ray SDK – 레코더에서 구성한 서비스 이름.
  - Amazon API Gateway – *api-name*/*stage*.
- 서비스 유형(문자열) – 트레이스 맵에 표시되는 서비스 유형입니다. X-Ray SDK의 경우 적절한 플러그인을 적용하여 서비스 유형을 설정합니다.
  - `AWS::ElasticBeanstalk::Environment` - AWS Elastic Beanstalk 환경(플러그인).
  - `AWS::EC2::Instance` – Amazon EC2 인스턴스 (플러그인).
  - `AWS::ECS::Container` – 아마존 ECS 컨테이너 (플러그인).
  - `AWS::APIGateway::Stage` – Amazon API Gateway 단계
  - `AWS::AppSync::GraphQLAPI` - AWS AppSync API 요청입니다.
- 호스트(문자열) - HTTP 호스트 헤더에 있는 호스트 이름.
- HTTP 메서드(문자열) – HTTP 요청 메서드.
- URL 경로(문자열) - 요청의 URL 경로.
  - X-Ray SDK – HTTP 요청 URL의 경로 부분.

- 리소스 ARN(문자열) - 서비스를 실행하는 AWS 리소스의 ARN입니다.
  - X-Ray SDK – 지원되지 않습니다. SDK는 리소스 ARN이 *로 설정된 규칙만 사용할 수 있습니다.
  - Amazon API Gateway – 스테이지 ARN.
- (선택 사항) 속성(키와 값) – 샘플링 결정을 내릴 때 알려진 세그먼트 속성.
  - X-Ray SDK – 지원되지 않습니다. SDK는 속성을 지정하는 규칙을 무시합니다.
  - Amazon API Gateway – 원본 HTTP 요청의 헤더.

샘플링 규칙 예

Example – 리저버가 없고 비율이 낮은 기본 규칙

기본 규칙의 리저버와 비율을 수정할 수 있습니다. 기본 규칙은 다른 규칙과 일치하지 않는 요청에 적용됩니다.

- 리저버: **0**
- 속도: **5** (JSON 문서를 사용하여 구성한 경우 **0.05**)

Example – 문제가 있는 경로에 대한 모든 요청을 추적하는 디버깅 규칙

디버깅을 위해 우선 순위가 높은 규칙이 일시적으로 적용되었습니다.

- 규칙 이름: **DEBUG - history updates**
- 우선 순위: **1**
- 리저버: **1**
- 속도: **100** (JSON 문서를 사용하여 구성한 경우 **1**)
- 서비스 이름: **Scorekeep**
- 서비스 유형: *****
- 호스트: *****
- HTTP 메서드: **PUT**
- URL 경로: **/history/***
- 리소스 ARN: *****

Example – POST에 대한 더 높은 최소 비율

- 규칙 이름: **POST minimum**

- 우선 순위: **100**
- 리저버: **10**
- 속도: **10** (JSON 문서를 사용하여 구성한 경우 **.1**)
- 서비스 이름: *
- 서비스 유형: *
- 호스트: *
- HTTP 메서드: **POST**
- URL 경로: *
- 리소스 ARN: *

샘플링 규칙을 사용하도록 서비스 구성

콘솔에서 구성한 샘플링 규칙을 사용하려면 X-Ray SDK에 추가 구성이 필요합니다. 샘플링 전략 구성에 대한 자세한 내용은 해당 언어의 구성 주제를 참조하십시오.

- Java: 샘플링 규칙
- Go: 샘플링 규칙
- Node.js: 샘플링 규칙
- Python: 샘플링 규칙
- Ruby: 샘플링 규칙
- .NET: 샘플링 규칙

API 게이트웨이의 경우 에 대한 Amazon API Gateway 활성 추적 지원 AWS X-Ray 단원을 참조하십시오.

샘플링 결과 보기

X-Ray 콘솔 샘플링 페이지에는 서비스가 각 샘플링 규칙을 사용하는 방법에 대한 자세한 정보가 표시됩니다.

Trend(추세) 열에는 규칙이 지난 몇 분간 사용된 방법이 표시됩니다. 각 열에는 10초 동안의 통계가 표시됩니다.

샘플링 통계

- 총 일치된 규칙: 이 규칙과 일치한 요청 수입니다. 이 수에는 이 규칙과 일치했을 수 있었지만 먼저 우선 순위가 더 높은 규칙과 일치한 요청은 포함되지 않습니다.
- 총 샘플링 수: 기록된 요청 수입니다.
- 고정 비율로 샘플링됨: 규칙의 고정 비율을 적용하여 샘플링된 요청 수입니다.
- 리저버 한도로 샘플링됨: 가 할당한 할당량을 사용하여 샘플링된 요청 수입니다.
- 리저버에서 빌림: 리저버에서 빌려 샘플링된 요청 수입니다. 서비스에서 요청을 규칙에 처음 일치시킬 때 X-Ray가 아직 할당량을 할당하지 않았습니다. 그러나 리저버가 1이상인 경우 X-Ray가 할당량을 할당할 때까지 서비스는 초당 트레이스 하나를 빌립니다.

샘플링 통계와 서비스가 샘플링 규칙을 사용하는 방법에 대한 자세한 내용은 X-Ray API에 샘플링 규칙 사용 단원을 참조하십시오.

다음 단계

X-Ray API를 사용하여 샘플링 규칙을 관리할 수 있습니다. API를 사용하여 일정을 기반으로 또는 경보나 알림에 대한 응답으로 규칙을 프로그래밍 방식으로 생성하고 업데이트할 수 있습니다. 지침 및 추가 규칙 예제는 AWS X-Ray API로 샘플링, 그룹 및 암호화 설정 구성 단원을 참조하십시오.

또한 X-Ray SDK 및는 X-Ray API를 AWS 서비스 사용하여 샘플링 규칙을 읽고, 샘플링 결과를 보고하고, 샘플링 대상을 가져옵니다. 서비스는 각 규칙을 적용하는 빈도를 추적하고, 우선 순위에 따라 규칙을 평가하며, 요청이 X-Ray가 아직 서비스에 할당량을 할당하지 않은 규칙과 일치하는 경우 리저버에서 빌려야 합니다. 서비스가 샘플링에 API를 사용하는 방법에 대한 자세한 내용은 X-Ray API에 샘플링 규칙 사용 단원을 참조하십시오.

X-Ray SDK가 샘플링 API를 호출할 때, X-Ray 대몬(daemon)을 프록시로 사용합니다. TCP 포트 2000을 이미 사용하는 경우, 다른 포트에서 프록시를 실행하도록 데몬을 구성할 수 있습니다. 세부 정보는 AWS X-Ray 데몬 구성 섹션을 참조하세요.

콘솔 심층 연결

경로와 쿼리를 사용하여 특정 트레이스로 딥링크하거나 트레이스와 트레이스 맵의 필터링된 보기로 딥링크할 수 있습니다.

콘솔 페이지

- 시작 페이지 – xray/home#/welcome
- 시작하기 – xray/home#/getting-started

- 트레이스 맵 – xray/home#/service-map
- 트레이스 – xray/home#/traces

트레이스

개별 트레이스의 타임라인, 원시 및 맵 보기에서 링크를 생성할 수 있습니다.

트레이스 타임라인 – `xray/home#/traces/`*`trace-id`*

원시 트레이스 데이터 – `xray/home#/traces/`*`trace-id`*`/raw`

Example – 원시 트레이스 데이터

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

필터 표현식

필터링된 트레이스 목록으로 연결합니다.

필터링된 트레이스 보기 – `xray/home#/traces?filter=`*`filter-expression`*

Example – 필터 표현식

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

Example – 필터 표현식(URL로 인코딩됨)

```
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
```

필터 표현식에 대한 자세한 내용은 필터 표현식 사용 단원을 참조하십시오.

시간 범위

기간 또는 시작 및 종료 시간을 ISO8601 형식으로 지정합니다. 시간 범위는 UTC 기준이며 최대 6시간까지 가능합니다.

기간 – `xray/home#/`*`page`*`?timeRange=`*`range-in-minutes`*

Example – 지난 한 시간 동안의 트레이스 맵

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

시작 및 종료 시간 – xray/home#/*page*?timeRange=*start*~*end*

Example – 초 단위 시간 범위

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

Example – 분 단위 시간 범위

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

리전

해당 리전의 페이지에 AWS 리전 연결할을 지정합니다. 리전을 지정하지 않을 경우 콘솔이 마지막으로 방문한 리전으로 리디렉션됩니다.

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

Example – 미국 서부(오리건)(us-west-2)의 트레이스 맵

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

쿼리 파라미터가 다른 리전을 포함시킬 경우 리전 쿼리가 해시 앞에 오고 X-Ray 고유 쿼리가 페이지 이름 뒤에 옵니다.

Example – 미국 서부(오리건)(us-west-2)의 지난 한 시간 동안의 트레이스 맵

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

조합

Example – 최근 트레이스와 기간 필터

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

출력

- 페이지 – 트레이스
- 시간 범위 – 마지막 15분
- 필터 – duration >= 5 AND duration <= 8

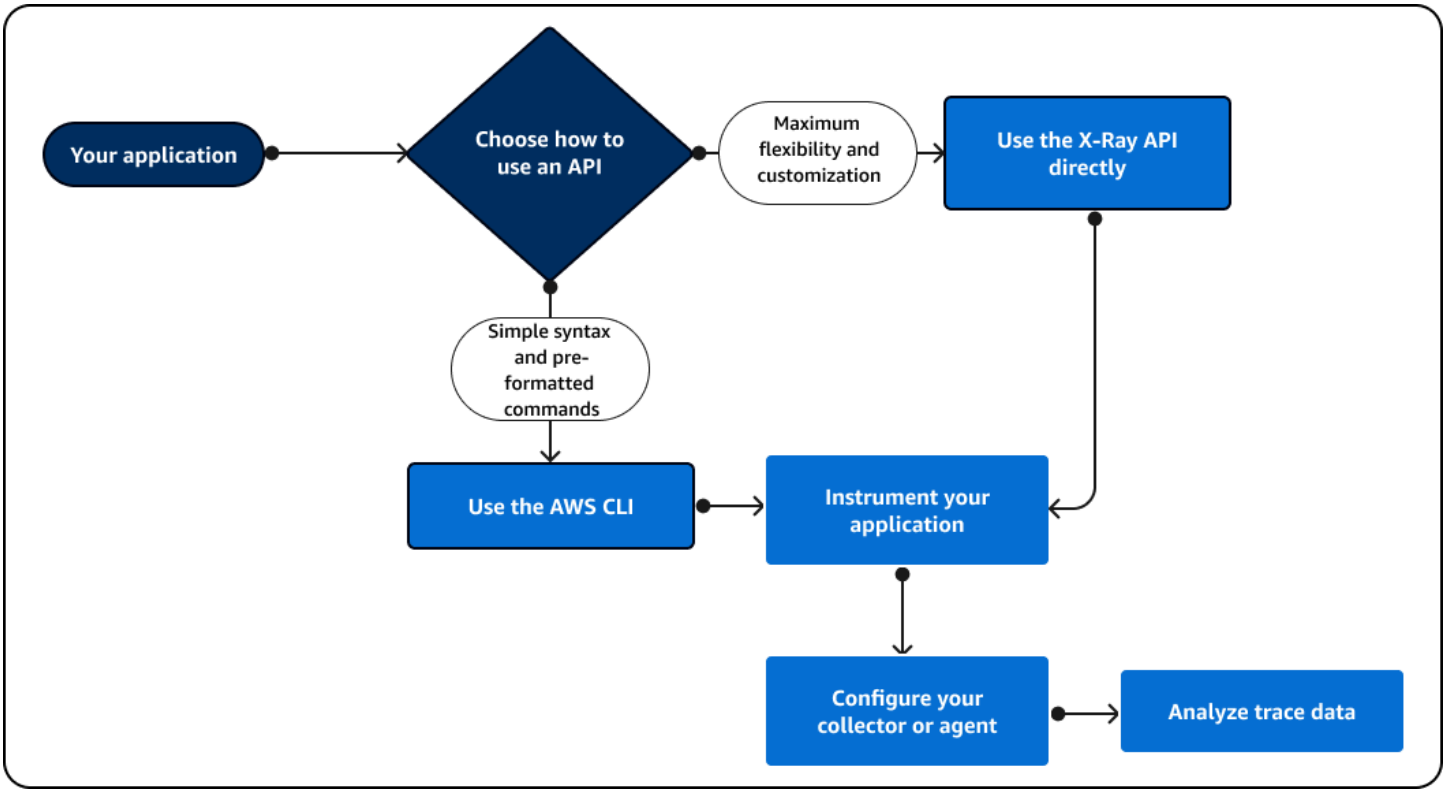
X-Ray API 사용

X-Ray SDK가 프로그래밍 언어를 지원하지 않는 경우 X-Ray APIs를 직접 사용하거나 AWS Command Line Interface (AWS CLI)를 사용하여 X-Ray API 명령을 호출할 수 있습니다. 다음 지침에 따라 API와 상호 작용하는 방법을 선택하세요.

- 사전 형식의 명령을 사용하거나 요청 내 옵션과 함께 보다 간단한 구문 AWS CLI 을 위해를 사용합니다.
- X-Ray에 대한 요청의 유연성과 사용자 지정을 극대화하려면 X-Ray API를 직접 사용합니다.

대신 X-Ray API를 직접 사용하는 경우 요청을 올바른 데이터 형식으로 파라미터화 AWS CLI해야 하며 인증 및 오류 처리를 구성해야 할 수도 있습니다.

다음 다이어그램은 X-Ray API와 상호 작용하는 방법을 선택하는 지침을 보여줍니다.



X-Ray API를 사용하여 트레이스 데이터를 X-Ray로 직접 전송합니다. X-Ray API는 다음과 같은 일반 작업을 포함하여 X-Ray SDK에서 사용할 수 있는 모든 함수를 지원합니다.

- PutTraceSegments - 세그먼트 문서를 X-Ray에 업로드합니다.
- BatchGetTraces - 트레이스 ID 목록에서 트레이스 목록을 검색합니다. 검색된 각 트레이스는 단일 요청의 세그먼트 문서 모음입니다.
- GetTraceSummaries - 트레이스에 대한 ID와 주석을 검색합니다. `FilterExpression`을 지정하여 트레이스 요약의 하위 집합을 검색할 수 있습니다.
- GetTraceGraph - 특정 트레이스 ID에 대한 서비스 그래프를 검색합니다.
- GetServiceGraph - 수신 요청을 처리하고 다운스트림 요청을 직접 호출하는 서비스를 설명하는 JSON 형식의 문서를 검색합니다.

애플리케이션 코드 내에서 AWS Command Line Interface (AWS CLI)를 사용하여 프로그래밍 방식으로 X-Ray와 상호 작용할 수도 있습니다. 는 다른 함수를 포함하여 X-Ray SDK에서 사용할 수 있는 모든 함수를 AWS CLI 지원합니다 AWS 서비스. 다음 함수는 앞에 나열된 API 작업의 간단한 형식 버전입니다.

- put-trace-segments - X-Ray에 세그먼트 문서를 업로드합니다.

- batch-get-traces - 트레이스 ID 목록에서 트레이스 목록을 검색합니다. 검색된 각 트레이스는 단일 요청의 세그먼트 문서 모음입니다.
- get-trace-summaries - 트레이스에 대한 ID와 주석을 검색합니다. `FilterExpression`을 지정하여 트레이스 요약의 하위 집합을 검색할 수 있습니다.
- get-trace-graph - 특정 트레이스 ID에 대한 서비스 그래프를 검색합니다.
- get-service-graph - 수신 요청을 처리하고 다운스트림 요청을 직접 호출하는 서비스를 설명하는 `JSON` 형식의 문서를 검색합니다.

시작하려면 운영 체제AWS CLI용를 설치해야 합니다. Linux, macOS 및 Windows 운영 체제를 AWS 지원합니다. X-Ray 명령 목록에 대한 자세한 내용은 X-RayAWS CLI 명령 참조 안내서를 참조하세요.

주제

- AWS CLI에서 AWS X-Ray API 사용
- AWS X-Ray로 추적 데이터 전송
- 에서 데이터 가져오기 AWS X-Ray
- AWS X-Ray API로 샘플링, 그룹 및 암호화 설정 구성
- X-Ray API에 샘플링 규칙 사용
- AWS X-Ray 세그먼트 문서

AWS CLI에서 AWS X-Ray API 사용

AWS CLI를 사용하면 X-Ray 서비스에 직접 액세스하고 X-Ray 콘솔이 서비스 그래프 및 원시 추적 데이터를 검색하는 데 사용하는 것과 동일한 APIs를 사용할 수 있습니다. 샘플 애플리케이션에는 AWS CLI에서 이러한 APIs를 사용하는 방법을 보여주는 스크립트가 포함되어 있습니다.

사전 조건

이 튜토리얼에서는 Scorekeep 샘플 애플리케이션과 포함된 스크립트를 사용하여 트레이스 데이터와 서비스 맵을 생성합니다. 튜토리얼 시작하기의 지침에 따라 애플리케이션을 시작합니다.

이 자습서에서는 AWS CLI 를 사용하여 X-Ray API의 기본 사용을 보여줍니다. Windows, Linux 및 OS-X에 사용할 수 있는 AWS CLI는 모든에 대해 퍼블릭 APIs에 대한 명령줄 액세스를 제공합니다 AWS 서비스.

> **Note**
>
> Scorekeep 샘플 애플리케이션이 생성된 리전과 동일한 리전으로 AWS CLI 가 구성되어 있는지 확인해야 합니다.

샘플 애플리케이션 테스트용으로 포함된 스크립트는 cURL을 사용하여 API 및 `jq`로 트래픽을 보내 출력을 구문 분석합니다. `jq` 실행 파일은 stedolan.github.io에서 다운로드할 수 있고, `curl` 실행 파일은 https://curl.haxx.se/download.html에서 다운로드할 수 있습니다. 대부분의 Linux 및 OS X 설치에는 cURL이 포함되어 있습니다.

데이터 추적 생성

웹 앱은 게임이 진행되는 동안 몇 초마다 계속해서 API로 가는 트래픽을 생성하지만 생성되는 요청은 한 가지 유형뿐입니다. API를 테스트하는 동안 `test-api.sh` 스크립트를 사용하여 종단 간 시나리오를 실행하고 보다 다양한 트레이스 데이터를 생성합니다.

`test-api.sh` 스크립트를 사용하려면

1. Elastic Beanstalk 콘솔을 엽니다.
2. 사용 중인 환경의 관리 콘솔로 이동합니다.
3. 페이지 헤더에서 환경 URL을 복사합니다.
4. `bin/test-api.sh`를 열고 API 값을 환경 URL로 바꿉니다.

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

5. API로 가는 트래픽을 생성하는 스크립트를 실행합니다.

```
~/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
```

X-Ray API 사용

AWS CLI는 GetServiceGraph 및를 포함하여 X-Ray가 제공하는 모든 API 작업에 대한 명령을 제공합니다GetTraceSummaries. 지원되는 모든 작업과 각 작업에서 사용되는 데이터 형식에 대한 자세한 내용은 AWS X-Ray API 참조를 참조하십시오.

Example bin/service-graph.sh

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

이 스크립트는 마지막 10분간의 서비스 그래프를 가져옵니다.

```
~/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'
```

이 스크립트는 1분 전과 2분 전 사이에 생성된 트레이의 URL을 가져옵니다.

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

이 스크립트는 1분 전과 2분 전 사이에 생성된 모든 트레이를 가져옵니다.

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

정리

Elastic Beanstalk 환경을 종료하여 Amazon EC2 인스턴스, DynamoDB 테이블 및 기타 리소스를 종료합니다.

Elastic Beanstalk 환경을 종료하려면

1. Elastic Beanstalk 콘솔을 엽니다.

2. 사용 중인 환경의 관리 콘솔로 이동합니다.
3. 작업을 선택합니다.
4. [Terminate Environment]를 선택합니다.
5. 종료를 선택합니다.

트레이스 데이터는 30일 후에 X-Ray에서 자동으로 삭제됩니다.

AWS X-Ray로 추적 데이터 전송

트레이스 데이터를 세그먼트 문서 형식으로 X-Ray로 전송할 수 있습니다. 세그먼트 문서는 애플리케이션이 요청을 처리하기 위해 수행하는 작업에 대한 정보를 포함하는 JSON 형식 문자열입니다. 애플리케이션은 스스로 수행하는 작업에 대한 데이터를 세그먼트에 기록하거나 다운스트림 서비스 및 리소스를 사용하는 작업을 하위 세그먼트에 기록할 수 있습니다.

세그먼트는 애플리케이션이 수행하는 작업에 대한 정보를 기록합니다. 세그먼트는 적어도 작업에 소비된 시간, 이름 및 두 개의 ID를 기록합니다. 트레이스 ID는 서비스 간에 이동하는 요청을 트레이스합니다. 세그먼트 ID는 단일 서비스의 요청에 대해 완료된 작업을 트레이스합니다.

Example 최소 완료 세그먼트

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

요청이 수신되면 요청이 완료될 때까지 진행률 세그먼트를 자리 표시자로 전송할 수 있습니다.

Example 진행 중인 세그먼트

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

```
}
```

세그먼트를 직접, `PutTraceSegments`, 또는 X-Ray 대몬(daemon)을 통해 X-Ray로 보낼 수 있습니다.

대부분의 애플리케이션은 AWS SDK를 사용하여 다른 서비스를 호출하거나 리소스에 액세스합니다. 다운스트림 호출에 대한 정보를 하위 세그먼트에 기록합니다. X-Ray는 하위 세그먼트를 사용하여 세그먼트를 전송하지 않고 서비스 그래프에 세그먼트에 대한 항목을 생성하지 않는 다운스트림 서비스를 식별합니다.

하위 세그먼트는 전체 세그먼트 문서에 임베드되거나 개별적으로 전송될 수 있습니다. 하위 세그먼트를 개별적으로 전송하여 장시간 실행되는 요청에 대한 다운스트림 호출을 비동기식으로 트레이스하거나 최대 세그먼트 문서 크기(64 kB)를 초과하지 않도록 합니다.

Example 하위 세그먼트

하위 세그먼트는 `subsegment`의 `type`과 상위 세그먼트를 식별하는 `parent_id`를 갖습니다.

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

세그먼트와 하위 세그먼트에 포함할 수 있는 필드 및 값에 대한 자세한 내용은 AWS X-Ray 세그먼트 문서 단원을 참조하십시오.

Sections

- 추적 ID 생성
- PutTraceSegments 사용
- 세그먼트 문서를 X 대몬(daemon)으로 전송

추적 ID 생성

데이터를 X-Ray로 전송하려면 각 요청에 대해 고유한 트레이스 ID를 생성해야 합니다.

X-Ray 트레이스 ID 형식

X-Ray `trace_id`는 하이픈으로 구분된 3개의 숫자로 구성됩니다. 예: `1-58406520-a006649127e371903a2de979`. 여기에는 다음이 포함됩니다.

- 버전 번호(1)
- 원래 요청의 시간(8자리 16진수를 사용하는 Unix 에포크 시간)

  예를 들어, 에포크 시간으로 2016년 12월 1일 오전 10시(PST)는 `1480615200`초이며, 16진수로는 `58406520`입니다.
- 트레이스의 96비트 글로벌 고유 식별자(24자리 16진수)

> **Note**
>
> X-Ray는 이제 OpenTelemetry 및 W3C 트레이스 컨텍스트 사양을 준수하는 기타 프레임워크를 사용하여 생성된 트레이스 ID를 지원합니다. X-Ray로 전송할 때 W3C 트레이스 ID는 X-Ray 트레이스 ID 형식이어야 합니다. 예를 들어, W3C 트레이스 ID `4efaaf4d1e8720b39541901950019ee5`는 X-Ray로 전송할 때 `1-4efaaf4d-1e8720b39541901950019ee5`와 같은 형식이어야 합니다. X-Ray 트레이스 ID에는 원래 요청 타임스탬프(Unix 에포크 시간)가 포함되어 있지만, W3C 트레이스 ID를 X-Ray 형식으로 전송할 때는 필요하지 않습니다.

테스트용 엑스레이 트레이스 ID를 생성하는 스크립트를 작성할 수 있습니다. 다음은 두 가지 예입니다.

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"
```

트레이스 ID를 생성하고 세그먼트를 X-Ray 대몬(daemon)으로 보내는 스크립트에 대해 Scorekeep 샘플 애플리케이션을 확인합니다.

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

PutTraceSegments 사용

PutTraceSegments API를 사용하여 세그먼트 문서를 업로드할 수 있습니다. 이 API에는 JSON 세그먼트 문서의 목록을 가져오는 단일 파라미터 `TraceSegmentDocuments`가 있습니다.

AWS CLI에서 `aws xray put-trace-segments` 명령을 사용하여 세그먼트 문서를 X-Ray로 직접 전송합니다.

```
$ 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 명령 처리기와 Windows PowerShell은 JSON 문자열에서의 인용 부호 요구 사항이 다릅니다. 자세한 내용은 AWS CLI 사용 설명서의 문자열 인용을 참조하십시오.

출력에는 처리를 실패한 세그먼트가 모두 나열됩니다. 예를 들어 트레이스 ID의 날짜가 너무 먼 과거일 경우 다음과 같은 오류가 반환됩니다.

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

```
        }
    ]
}
```

여러 세그먼트 문서를 공백으로 구분하여 동시에 전달할 수 있습니다.

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

세그먼트 문서를 X 대몬(daemon)으로 전송

세그먼트 문서를 API로 전송하는 대신, 세그먼트 및 하위 세그먼트를 X-Ray 대몬(daemon)으로 전송할 수 있습니다. 이 대몬(daemon)은 세그먼트 및 하위 세그먼트를 버퍼링하다가 배치 단위로 X-Ray API로 업로드합니다. X-Ray SDK는 AWS 로 직접 호출을 방지하기 위해 세그먼트 문서를 대몬(daemon)으로 전송합니다.

> **Note**
>
> 데몬을 실행하는 방법은 로컬에서 X-Ray 대몬(daemon) 실행하기 단원을 참조하십시오.

데몬 헤더 `{"format": "json", "version": 1}\n`가 앞에 붙은 JSON 형식 세그먼트를 UDP 포트 2000으로 전송합니다.

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

Linux에서는 세그먼트 문서를 Bash 터미널에서 데몬으로 전송할 수 있습니다. 헤더와 세그먼트 문서를 텍스트 파일로 저장하고 `cat`을 사용하여 `/dev/udp`로 파일을 파이프합니다.

```
$ 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"}
```

대몬(daemon) 로그를 통해 세그먼트가 X-Ray로 전송되었는지 확인합니다.

```
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)
```

에서 데이터 가져오기 AWS X-Ray

AWS X-Ray 는 사용자가 전송하는 추적 데이터를 처리하여 전체 추적, 추적 요약 및 서비스 그래프를 JSON으로 생성합니다. AWS CLI를 사용하여 API에서 직접 생성된 데이터를 검색할 수 있습니다.

Sections

서비스 그래프 가져오기

GetServiceGraph API를 사용하여 JSON 서비스 그래프를 검색할 수 있습니다. API는 시작 시간과 종료 시간을 필요로 합니다. 이 시간은 Linux에서 `date` 명령을 사용하여 계산할 수 있습니다.

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

`date +%s`는 날짜를 초 단위로 출력합니다. 이 숫자를 종료 시간으로 사용하고, 여기에서 시간을 차감하여 시작 시간을 구합니다.

Example 마지막 10분간의 서비스 그래프를 가져오기 위한 스크립트

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

다음 예제는 클라이언트 노드, EC2 인스턴스, DynamoDB 테이블 및 Amazon SNS 주제를 포함한 4개의 노드가 있는 서비스 그래프를 보여줍니다.

Example GetServiceGraph 출력

```
{
    "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
                }
```

```
            ]
        }
    ]
}
```

그룹별 서비스 그래프 검색

그룹의 콘텐츠에 기반한 서비스 그래프를 호출하려면 groupName 또는 groupARN를 포함합니다. 다음의 예는 Example1이라는 그룹에 서비스 그래프를 호출하는 방법을 보여줍니다.

Example 그룹 Example1의 이름으로 서비스 그래프를 검색하기 위한 스크립트

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

추적 검색

GetTraceSummaries API를 사용하여 트레이스 요약의 목록을 가져올 수 있습니다. 트레이스 요약에는 주석, 요청/응답 정보 및 ID를 포함한 전체 트레이스를 다운로드하려는 트레이스를 식별하는 데 사용할 수 있는 정보가 포함됩니다.

`aws xray get-trace-summaries`를 호출할 때 사용할 수 있는 두 가지 TimeRangeType 플래그가 있습니다.

- TraceID — 기본 `GetTraceSummaries` 검색은 TraceID 시간을 사용하며 계산된 `[start_time, end_time)` 범위 내에서 시작된 트레이스를 반환합니다. 타임스탬프의 범위는 TraceID 내의 타임스탬프 인코딩을 기반으로 계산되거나 수동으로 정의할 수 있습니다.
- Event 시간 - AWS X-Ray는 시간 경과에 따라 발생하는 이벤트를 검색하기 위해 이벤트 타임스탬프를 사용해 트레이스를 검색할 수 있습니다. Event 시간은 트레이스가 시작된 시간에 관계없이 `[start_time, end_time)` 범위 동안 활성 트레이스를 반환합니다.

`aws xray get-trace-summaries` 명령을 사용하여 트레이스 요약의 목록을 가져옵니다. 다음 명령은 기본 TraceId 시간을 사용하여 지난 1분에서 2분 사이의 트레이스 요약 목록을 가져옵니다.

Example 트레이스 요약을 획득하는 스크립트

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

Example GetTraceSummaries 출력

```
{
    "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
}
```

BatchGetTraces API를 사용하여 출력 내 트레이스 ID로 전체 트레이스를 검색합니다.

Example BatchGetTraces 명령

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

Example BatchGetTraces 출력

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


전체 트레이스에는 동일한 트레이스 ID로 수신된 모든 세그먼트 문서로부터 컴파일된 각 세그먼트의 문서가 포함됩니다. 데이터는 애플리케이션에서 X-Ray로 전송되므로 이러한 문서는 데이터를 나타내지 않습니다. 그 대신 X-Ray; 서비스에 의해 생성된 처리된 문서를 나타냅니다. X-Ray는 애플리케이션이 전송한 세그먼트 문서를 컴파일하고 세그먼트 문서 스키마를 준수하지 않는 데이터를 제거하여 전체 트레이스 문서를 생성합니다.

또한 X-Ray는 세그먼트를 직접 전송하지 않는 서비스에 대한 다운스트림 호출을 위해 추정된 세그먼트를 생성합니다. 예를 들어, 계측되는 클라이언트를 사용하여 DynamoDB를 직접 호출하는 경우, X-Ray SDK가 자신의 관점에서 직접 호출에 대한 세부 정보를 하위 세그먼트에 기록합니다. 하지만 DynamoDB는 해당 세그먼트를 전송하지 않습니다. X-Ray는 하위 세그먼트의 정보를 사용하여 추론된 세그먼트를 생성하여 트레이스 맵에 DynamoDB 리소스를 나타내고 이를 트레이스 문서에 추가합니다.

API에서 여러 트레이스를 가져오려면 AWS CLI 쿼리를 통해 `get-trace-summaries` 출력에서 추출할 수 있는 트레이스 ID 목록이 필요합니다. 특정 기간에 대해 전체 트레이스를 가져오려면 목록을 `batch-get-traces`의 입력으로 리디렉션합니다.

Example 1분간의 전체 트레이스를 가져오기 위한 스크립트

```
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[*]'
```

근본 원인 분석 가져오기 및 구체화

GetTraceSummaries API로 트레이스 요약이 생성되면 바로 부분 트레이스 요약을 JSON 형식으로 다시 사용하여 근본 원인에 따라 구체화된 필터 표현식을 생성할 수 있습니다. 구체화 단계에 대한 연습은 아래 예제를 참조하십시오.

Example GetTraceSummaries 출력 예제 - 응답 시간 근본 원인 섹션

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

위 출력을 편집하여 생략하면 일치하는 근본 원인 개체마다 이 JSON을 필터로 사용할 수 있습니다. JSON의 필드는 모든 후보 일치가 정확해야 합니다. 그렇지 않으면 트레이스가 반환되지 않습니다. 제거된 필드는 필터 표현식 쿼리 구조와 호환되는 형식인 와일드카드 값이 됩니다.

Example 변경된 응답 시간 근본 원인

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

```
}
```

이 JSON은 `rootcause.json = #[{}]` 호출을 통해 필터 표현식의 일부로 사용됩니다. 필터 표현식을 사용해 쿼리하는 방법에 대한 자세한 내용은 필터 표현식 단원을 참조하십시오.

Example JSON 필터 예제

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

AWS X-Ray API로 샘플링, 그룹 및 암호화 설정 구성

AWS X-Ray 는 샘플링 규칙, 그룹 규칙 및 암호화 설정을 구성하기 위한 APIs 제공합니다.

Sections

- 암호화 설정
- 샘플링 규칙
- Groups

암호화 설정

PutEncryptionConfig를 사용하여 암호화에 사용할 AWS Key Management Service (AWS KMS) 키를 지정합니다.

> **Note**
>
> X-Ray는 비대칭 KMS 키를 지원하지 않습니다.

```
$ 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"
    }
```

```
}
```

키 ID로는 예제에 나온 것처럼 별칭이나 키 ID, 또는 Amazon 리소스 이름(ARN)을 사용할 수 있습니다.

GetEncryptionConfig를 사용하여 현재 구성을 가져옵니다. X-Ray가 설정 적용을 마치면 상태는 UPDATING에서 ACTIVE로 바뀝니다.

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

KMS 키 사용을 중단하고 기본 암호화를 사용하려면 암호화 유형을 NONE으로 설정합니다.

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

샘플링 규칙

X-Ray API로 계정의 샘플링 규칙을 관리할 수 있습니다. 태그 추가 및 관리에 대한 자세한 내용은 X-Ray 샘플링 규칙 및 그룹 태그 지정하기 섹션을 참조하세요.

GetSamplingRules로 모든 샘플링 규칙을 가져옵니다.

```
$ 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
        }
    ]
}
```

기본 규칙은 다른 규칙과 일치하지 않는 모든 요청에 적용됩니다. 우선 순위가 가장 낮은 규칙으로 삭제할 수 없습니다. 그러나 UpdateSamplingRule로 비율과 리저버 크기를 변경할 수 있습니다.

Example **UpdateSamplingRule**에 대한 API 입력 – 10000-default.json

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

다음 예제에서는 이전 파일을 입력으로 사용하여 기본 규칙을 리저버가 없는 1퍼센트로 변경합니다. 태그는 선택 사항입니다. 태그를 추가하는 경우 태그 키가 필요하며 태그 값은 선택 사항입니다. 샘플링 규칙에서 기존 태그를 제거하려면 UntagResource를 사용하세요.

```
$ 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
        },
```

CreateSamplingRule로 추가 샘플링 규칙을 만듭니다. 규칙을 만들 때 대부분의 규칙 필드는 필수입니다. 다음 예제에서는 규칙 두 개를 만듭니다. 이 첫 번째 규칙은 Scorekeep 샘플 애플리케이션의 기본 비율을 설정하며, 우선 순위가 더 높은 규칙과 일치하지 않는 API에서 처리하는 모든 요청과 일치합니다.

Example **UpdateSamplingRule**에 대한 API 입력 – 9000-base-scorekeep.json

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

두 번째 규칙은 Scorekeep에도 적용되지만, 우선 순위가 더 높고 구체적입니다. 이 규칙은 폴링 요청에 대해 매우 낮은 샘플링 비율을 설정합니다. 게임 상태에 변경 사항이 있는지 확인하기 위해 클라이언트가 몇 초마다 수행하는 GET 요청입니다.

Example **UpdateSamplingRule**에 대한 API 입력 – 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
    }
}
```

태그는 선택 사항입니다. 태그를 추가하는 경우 태그 키가 필요하며 태그 값은 선택 사항입니다.

```
$ 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
    }
}
```

샘플링 규칙을 삭제하려면 DeleteSamplingRule을 사용합니다.

```
$ 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
    }
}
```

Groups

X-Ray API를 이용해 계정의 그룹을 관리할 수 있습니다. 그룹은 필터 표현식으로 정의한 추적 모음입니다. 그룹을 사용하여 추가 서비스 그래프를 작성하고 Amazon CloudWatch 지표를 제공할 수 있습니다. X-Ray API를 통한 서비스 그래프 및 지표 작업에 대한 자세한 내용은 에서 데이터 가져오기 AWS X-Ray 항목을 참조하십시오. 그룹에 대한 자세한 정보는 그룹 구성 섹션을 참조하세요. 태그 추가 및 관리에 대한 자세한 내용은 X-Ray 샘플링 규칙 및 그룹 태그 지정하기 섹션을 참조하세요.

`CreateGroup`이(가) 있는 그룹 생성 태그는 선택 사항입니다. 태그를 추가하는 경우 태그 키가 필요하며 태그 값은 선택 사항입니다.

```
$ 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}"
}
```

`GetGroups`로 기존 그룹을 모두 가져옵니다.

```
$ 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"
}
```

UpdateGroup이(가) 있는 그룹 업데이트 태그는 선택 사항입니다. 태그를 추가하는 경우 태그 키가 필요하며 태그 값은 선택 사항입니다. 그룹에서 기존 태그를 제거하려면 UntagResource를 사용하세요.

```
$ 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}"
}
```

DeleteGroup이(가) 있는 그룹 삭제

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

X-Ray API에 샘플링 규칙 사용

AWS X-Ray SDK는 X-Ray API를 사용하여 샘플링 규칙을 가져오고, 샘플링 결과를 보고하고, 할당량을 가져옵니다. 이러한 API를 사용하여 샘플링 규칙의 작동 방식을 더 잘 이해하거나, X-Ray SDK가 지원하지 않는 언어로 샘플링을 구현할 수 있습니다.

먼저 GetSamplingRules로 모든 샘플링 규칙을 가져옵니다.

```
$ 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
        }
    ]
}
```

출력에는 기본 규칙과 사용자 지정 규칙이 포함됩니다. 아직 샘플링 규칙을 만들지 않은 경우 샘플링 규칙을 참조하십시오.

우선 순위의 오름차순에 따라 수신 요청과 비교하여 규칙을 평가합니다. 규칙이 일치하면 고정 비율과 리저버 크기를 사용하여 샘플링 결정을 내립니다. 샘플링된 요청을 기록하고 샘플링되지 않은 요청을 무시합니다(추적 목적). 샘플링 결정을 내릴 때 규칙 평가를 중지합니다.

규칙 리저버 크기는 고정 비율을 적용하기 전에 초당 기록할 목표 트레이스 수입니다. 리저버는 모든 서비스에 누적 적용되므로 직접 사용할 수 없습니다. 하지만 0이 아닌 경우 X-Ray가 할당량을 할당할 때까지 리저버에서 초당 트레이스 하나를 빌릴 수 있습니다. 할당량을 수신하기 전에 매초 첫 번째 요청을 기록하고, 추가 요청에 고정 비율을 적용합니다. 고정 비율은 0~1.00(100%)의 십진수입니다.

다음 예제에서는 지난 10초간 내린 샘플링 결정에 대한 세부 정보가 포함된 GetSamplingTargets에 대한 호출을 보여 줍니다.

```
$ 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-Ray의 응답에는 리저버에서 빌리는 대신에 사용할 할당량이 포함됩니다. 이 예제에서 서비스는 10초간 리저버에서 트레이스 10개를 빌렸고, 다른 100개 요청에 고정 비율 10%를 적용했으며, 그 결과 샘플링된 요청은 총 20개입니다. 할당량은 5분 (유효 시간으로 표시) 동안 또는 새 할당량이 할당될 때까지 유효합니다. X-Ray는 기본값보다 긴 보고 간격을 할당할 수 있지만 여기서는 그렇지 않습니다.

> **Note**
>
> X-Ray의 응답에는 처음 직접 호출할 때의 할당량이 포함되지 않을 수 있습니다. 할당량이 할당될 때까지 리저버에서 계속 빌리십시오.

응답의 다른 두 필드는 입력과 관련된 문제를 나타낼 수 있습니다. [GetSamplingRules](#)를 마지막으로 호출한 시간과 비교하여 `LastRuleModification`을 확인합니다. 더 최신인 경우 규칙의 새 사본을 가져옵니다. `UnprocessedStatistics`에는 규칙이 삭제되었거나, 입력의 통계 문서가 너무 오래되었음을 나타내는 오류 또는 권한 오류가 포함될 수 있습니다.

AWS X-Ray 세그먼트 문서

트레이스 세그먼트는 애플리케이션이 처리하는 요청의 JSON 표현입니다. 트레이스 세그먼트는 원래 요청에 대한 정보, 애플리케이션이 로컬에서 수행하는 작업에 대한 정보, 애플리케이션이 AWS 리소스, HTTP APIs 및 SQL 데이터베이스에 대해 수행하는 다운스트림 호출에 대한 정보가 포함된 하위 세그먼트를 기록합니다.

세그먼트 문서는 세그먼트 정보를 X-Ray에 전달합니다. 세그먼트 문서는 최대 64kB 까지 가능하며, 하위 세그먼트와 요청이 진행 중임을 표시하는 세그먼트 조각 또는 별도로 전송된 단일 하위 세그먼트를 포함한 전체 세그먼트가 있습니다. `PutTraceSegments` API를 사용하여 X-Ray에 세그먼트 문서를 직접 보낼 수 있습니다.

X-Ray는 세그먼트 문서를 컴파일하고 처리해 각각 `GetTraceSummaries` 및 `BatchGetTraces` API로 액세스할 수 있는 쿼리 가능한 트레이스 요약과 전체 트레이스를 생성합니다. X-Ray로 전송할 수 있는 세그먼트와 하위 세그먼트 외에도, 이 서비스는 하위 세그먼트의 정보를 이용해 추론 세그먼트를 생성하고 전체 트레이스에 추가합니다. 추론된 세그먼트는 트레이스 맵에서 다운스트림 서비스와 리소스를 나타냅니다.

X-Ray는 세그먼트 문서를 위한 JSON 스키마를 제공합니다. 스키마는 xray-segmentdocument-schema-v1.0.0에서 다운로드할 수 있습니다. 스키마에 나열된 필드와 객체는 다음 단원에서 더 자세히 설명합니다.

세그먼트 필드의 서브셋은 필터 표현식에 사용할 수 있도록 X-Ray로 인덱스됩니다. 예를 들어 세그먼트의 `user` 필드를 고유 식별자로 설정하면, X-Ray 콘솔에서 또는 `GetTraceSummaries` API를 사용하여 특정 사용자와 관련된 세그먼트를 검색할 수 있습니다. 자세한 내용은 필터 표현식 사용 단원을 참조하십시오.

X-Ray SDK로 애플리케이션을 구성하면, SDK는 사용자를 위한 세그먼트 문서를 생성합니다. 세그먼트 문서를 X-Ray에 직접 전송하는 대신, SDK는 로컬 UDP 포트를 이용해 X-Ray 대몬(daemon)에 전송합니다. 자세한 내용은 세그먼트 문서를 X 대몬(daemon)으로 전송 단원을 참조하십시오.

Sections

- AWS 리소스 데이터
- 오류 및 예외
- SQL 쿼리

세그먼트 필드

세그먼트는 애플리케이션이 처리하는 요청에 대한 트레이싱 정보를 기록합니다. 세그먼트는 기본적으로 이름, ID, 시작 시간 및 트레이스 ID, 요청 종료 시간을 기록합니다.

Example 최소 완료 세그먼트

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

다음은 세그먼트에 필수적이거나 조건부 필수인 필드입니다.

> **Note**
>
> 특별한 언급이 없는 한 값은 문자열(최대 250자)이어야 합니다.

필수 세그먼트 필드

- `name` – 요청을 처리한 서비스의 논리명으로, 200자까지 가능합니다. 예를 들자면 애플리케이션 이름이나 도메인 이름이 있습니다. 이름에는 유니코드 문자, 숫자, 공백 및 다음 기호를 포함할 수 있습니다: `_`, `.`, `:`, `/`, `%`, `&`, `#`, `=`, `+`, `\`, `-`, `@`
- `id` – 동일한 트레이스 내 세그먼트에서 고유한 64비트 식별자 (16자리 16진수).
- `trace_id` – 단일 클라이언트 요청에서 생성된 모든 세그먼트 및 하위 세그먼트를 연결하는 고유한 식별자.

  X-Ray 트레이스 ID 형식

  X-Ray `trace_id`는 하이픈으로 구분된 3개의 숫자로 구성됩니다. 예: `1-58406520-a006649127e371903a2de979`. 여기에는 다음이 포함됩니다.

- 버전 번호(1)
- 원래 요청의 시간(8자리 16진수를 사용하는 Unix 에포크 시간)

  예를 들어, 에포크 시간으로 2016년 12월 1일 오전 10시(PST)는 1480615200초이며, 16진수로는 58406520입니다.
- 트레이스의 96비트 글로벌 고유 식별자(24자리 16진수)

> **Note**
>
> X-Ray는 이제 OpenTelemetry 및 W3C 트레이스 컨텍스트 사양을 준수하는 기타 프레임워크를 사용하여 생성된 트레이스 ID를 지원합니다. X-Ray로 전송할 때 W3C 트레이스 ID는 X-Ray 트레이스 ID 형식이어야 합니다. 예를 들어, W3C 트레이스 ID 4efaaf4d1e8720b39541901950019ee5는 X-Ray로 전송할 때 1-4efaaf4d-1e8720b39541901950019ee5와 같은 형식이어야 합니다. X-Ray 트레이스 ID에는 원래 요청 타임스탬프(Unix 에포크 시간)가 포함되어 있지만, W3C 트레이스 ID를 X-Ray 형식으로 전송할 때는 필요하지 않습니다.

> **트레이스 ID 보안**
>
> 트레이스 ID는 응답 헤더에 표시됩니다. 공격이 향후의 트레이스 ID를 계산하여 해당 ID로 애플리케이션에 요청을 보낼 수 없도록 보안 무작위 알고리즘을 사용하여 트레이스 ID를 생성합니다.

- `start_time` – 세그먼트가 생성된 시간을 에포크 시간으로 부동 소수점으로 표시한 숫자입니다. 예: 1480615200.010또는 1.480615200010E9. 소수점 자리는 필요한 만큼 사용하십시오. 가능하다면 마이크로초 해상도를 권장합니다.
- `end_time` – 세그먼트를 닫은 시간을 표시한 숫자입니다. 예: 1480615200.090또는 1.480615200090E9. `end_time` 또는 `in_progress`를 지정합니다.
- `in_progress` – 세그먼트가 시작되었지만 완료되지 않았음을 기록하고자 `end_time`을 지정하는 대신 `true`로 설정한 부울입니다. 애플리케이션이 처리하는 데 오래 걸리는 요청을 수신할 경우 처리 중 세그먼트를 전송하여 요청 수신을 트레이스합니다. 응답이 전송되면 완료 세그먼트를 전송하여 처리 중 세그먼트를 덮어씁니다. 요청 당 완성된 세그먼트 하나만 전송하며, 진행 중인 세그먼트는 1개 또는 0개입니다.

> **서비스 이름**
>
> 세그먼트의 name은 해당 세그먼트를 생성한 서비스의 도메인 이름 또는 논리적 이름과 일치해야 합니다. 하지만 이것이 적용되지는 않습니다. PutTraceSegments 허가를 받은 애플리케이션은 어떠한 이름을 사용하여도 세그먼트를 전송할 수 있습니다.

다음은 세그먼트의 선택 사항인 필드입니다.

선택형 세그먼트 필드

- service – 애플리케이션 정보가 있는 객체입니다.
  - version – 요청을 처리한 애플리케이션 버전을 표시하는 문자열입니다.
- user – 요청을 보낸 사용자를 식별하는 문자열입니다.
- origin - 애플리케이션을 실행하는 AWS 리소스 유형입니다.

  지원되는 값

  - AWS::EC2::Instance – Amazon EC2 인스턴스
  - AWS::ECS::Container – 아마존 ECS 컨테이너
  - AWS::ElasticBeanstalk::Environment – Elastic Beanstalk 환경

  애플리케이션에 여러 값을 적용할 수 있는 경우, 가장 구체적인 값을 사용하십시오. 예를 들어 Multicontainer Docker Elastic Beanstalk 환경은 Amazon ECS 컨테이너에서 애플리케이션을 실행하고, 이 컨테이너는 Amazon EC2 인스턴스에서 실행됩니다. 이 경우 환경이 다른 두 리소스의 상위 항목이므로 AWS::ElasticBeanstalk::Environment로 오리진을 설정합니다.
- parent_id – 요청이 구성한 애플리케이션에서 나오지 않았을 경우 사용자가 지정한 하위 세그먼트 ID입니다. X-Ray SDK는 다운스트림 HTTP 호출을 위해 상위 하위 세그먼트 ID를 트레이싱 헤더에 추가합니다. 중첩된 하위 세그먼트의 경우, 하나의 하위 세그먼트는 하나의 세그먼트를 포함하거나 상위 세그먼트로서 하위 세그먼트를 포함할 수 있습니다.
- http – 기존 HTTP 요청에 대한 정보가 있는 http 객체입니다.
- aws - 애플리케이션이 요청을 처리한 AWS 리소스에 대한 정보가 포함된 aws 객체입니다.
- error, throttle, fault, 및 cause – 오류가 발생했음을 표시하며 오류를 유발한 예외 사항에 대한 정보가 담긴 오류 필드입니다.
- annotations – X-Ray가 검색용으로 인덱스할 키-값 페어가 있는 annotations 객체입니다.
- metadata – 세그먼트에 저장할 추가 데이터가 있는 metadata 객체입니다.

- subsegments – subsegment 객체의 배열입니다.

하위 세그먼트

하위 세그먼트를 생성하여 AWS SDK, 내부 또는 외부 HTTP 웹 APIs에 대한 호출 또는 SQL 데이터베이스 쿼리를 통해 수행한 AWS 서비스 및 리소스에 대한 호출을 기록할 수 있습니다. 또한 디버그할 하위 세그먼트를 만들거나 애플리케이션의 코드 블록에 주석을 달 수도 있습니다. 하위 세그먼트는 다른 하위 세그먼트를 포함할 수 있기 때문에, 내부 함수 호출에 대한 메타데이터를 기록한 사용자 지정 하위 세그먼트는 다른 사용자 지정 하위 세그먼트와 다운스트림 호출을 위한 하위 세그먼트를 포함할 수 있습니다.

하위 세그먼트는 다운스트림 직접 호출을 생성한 서비스의 관점에서 해당 호출을 기록합니다. X-Ray는 하위 세그먼트를 사용하여 세그먼트를 전송하지 않고 서비스 그래프에 세그먼트에 대한 항목을 생성하지 않는 다운스트림 서비스를 식별합니다.

하위 세그먼트는 전체 세그먼트 문서에 임베드되거나 독립적으로 전송될 수 있습니다. 하위 세그먼트를 개별적으로 전송하여 장시간 실행되는 요청에 대한 다운스트림 호출을 비동기식으로 트레이스하거나 최대 세그먼트 문서 크기를 초과하지 않도록 합니다.

Example 임베디드 하위 세그먼트가 있는 세그먼트

독립 하위 세그먼트는 type의 subsegment과 상위 세그먼트를 식별하는 parent_id를 갖습니다.

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

장기 실행 요청이라면, 진행 중인 세그먼트를 전송하여 요청이 수신되었음을 X-Ray에 알린 후, 하위 세그먼트를 개별적으로 전송해 트레이스할 수 있게 한 다음 원래 요청을 완료해야 합니다.

Example 진행 중인 세그먼트

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

Example 독립 하위 세그먼트

독립 하위 세그먼트는 `type`의 `subsegment`, `trace_id`와 상위 세그먼트를 식별하는 `parent_id`를 갖습니다.

```
{
```

```
  "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
      }
  }
}
```

요청이 완료되면, `end_time`과 함께 다시 전송해 세그먼트를 닫으십시오. 완료된 세그먼트는 진행 중인 세그먼트를 덮어씁니다.

비동기 워크플로를 유발한 완료 요청에 하위 세그먼트를 개별적으로 전송할 수도 있습니다. 예를 들어 웹 API는 사용자가 요청한 작업을 시작하기 직전에 `OK 200` 응답을 반환하기도 합니다. 응답이 전송되는 즉시 전체 세그먼트를 X-Ray에 전송하고, 나중에 완료한 작업용 하위 세그먼트를 전송해도 됩니다. 세그먼트의 경우, 하위 세그먼트 조각을 하위 세그먼트가 시작된 기록으로 전송한 다음 다운스트림 호출이 완료되는 즉시 전체 하위 세그먼트로 덮어써도 됩니다.

다음은 하위 세그먼트에 필수적이거나 조건부 필수인 필드입니다.

> **Note**
>
> 특별한 언급이 없는 한 값은 최대 250자인 문자열이어야 합니다.

필수 하위 세그먼트 필드

- `id` – 동일한 트레이스 내 세그먼트에서 고유한 하위 세그먼트의 64비트 식별자입니다(16자리 16진수).

- name – 하위 세그먼트의 논리명입니다. 다운스트림 직접 호출의 경우에는 리소스나 서비스를 호출한 후 하위 세그먼트의 이름을 지정하십시오. 사용자 지정 하위 세그먼트의 경우에는 이를 구성한 코드 이름(예: 함수 이름)을 정한 후에 하위 세그먼트 이름을 지정하십시오.
- start_time – 하위 세그먼트가 생성된 시간을 에포크 시간으로 밀리초까지 부동 소수점으로 표시한 숫자입니다. 예: 1480615200.010또는 1.480615200010E9.
- end_time – 하위 세그먼트를 닫은 시간을 표시한 숫자입니다. 예: 1480615200.090또는 1.480615200090E9. end_time 또는 in_progress를 지정합니다.
- in_progress – 하위 세그먼트가 시작되었지만 완료되지 않았음을 기록하고자 end_time을 지정하는 대신 true로 설정한 부울입니다. 다운스트림 요청 당 완성된 하위 세그먼트 하나만 전송하며, 진행 중인 하위 세그먼트는 1개 또는 0개입니다.
- trace_id – 하위 세그먼트의 상위 세그먼트 트레이스 ID입니다. 하위 세그먼트를 개별적으로 전송할 때만 필요합니다.

  X-Ray 트레이스 ID 형식

  X-Ray trace_id는 하이픈으로 구분된 3개의 숫자로 구성됩니다. 예: 1-58406520-a006649127e371903a2de979. 여기에는 다음이 포함됩니다.

  - 버전 번호(1)
  - 원래 요청의 시간(8자리 16진수를 사용하는 Unix 에포크 시간)

    예를 들어, 에포크 시간으로 2016년 12월 1일 오전 10시(PST)는 1480615200초이며, 16진수로는 58406520입니다.
  - 트레이스의 96비트 글로벌 고유 식별자(24자리 16진수)

  > **Note**
  >
  > X-Ray는 이제 OpenTelemetry 및 W3C 트레이스 컨텍스트 사양을 준수하는 기타 프레임워크를 사용하여 생성된 트레이스 ID를 지원합니다. X-Ray로 전송할 때 W3C 트레이스 ID는 X-Ray 트레이스 ID 형식이어야 합니다. 예를 들어, W3C 트레이스 ID 4efaaf4d1e8720b39541901950019ee5는 X-Ray로 전송할 때 1-4efaaf4d-1e8720b39541901950019ee5와 같은 형식이어야 합니다. X-Ray 트레이스 ID에는 원래 요청 타임스탬프(Unix 에포크 시간)가 포함되어 있지만, W3C 트레이스 ID를 X-Ray 형식으로 전송할 때는 필요하지 않습니다.

- `parent_id` – 하위 세그먼트의 상위 세그먼트에 적용된 세그먼트 ID입니다. 하위 세그먼트를 개별적으로 전송할 때만 필요합니다. 중첩된 하위 세그먼트의 경우, 하나의 하위 세그먼트는 하나의 세그먼트를 포함하거나 상위 세그먼트로서 하위 세그먼트를 포함할 수 있습니다.
- `type` – `subsegment`. 하위 세그먼트를 개별적으로 전송할 때만 필요합니다.

다음은 하위 세그먼트의 선택 사항인 필드입니다.

선택형 하위 세그먼트 필드

- `namespace` – AWS SDK 호출의 경우 `aws`이며, 다른 다운스트림 호출의 경우 `remote`입니다.
- `http` – 발신 HTTP 호출에 대한 정보가 있는 `http` 객체입니다.
- `aws` - 애플리케이션이 호출한 다운스트림 AWS 리소스에 대한 정보가 포함된 `aws` 객체입니다.
- `error`, `throttle`, `fault`, 및 `cause` – 오류가 발생했음을 표시하며 오류를 유발한 예외 사항에 대한 정보가 담긴 오류 필드입니다.
- `annotations` – X-Ray가 검색용으로 인덱스할 키-값 페어가 있는 `annotations` 객체입니다.
- `metadata` – 세그먼트에 저장할 추가 데이터가 있는 `metadata` 객체입니다.
- `subsegments` – `subsegment` 객체의 배열입니다.
- `precursor_ids` – 이 하위 세그먼트 이전에 완료한 상위 세그먼트를 식별하는 하위 세그먼트 ID 배열입니다.

HTTP 요청 데이터

HTTP 블록을 이용해 애플리케이션이 처리한 HTTP 요청 정보를 (세그먼트에) 기록하거나 애플리케이션이 다운스트림 HTTP API에 적용한 HTTP 요청 정보를 (하위 세그먼트에) 기록합니다. 이 객체에 있는 필드 대부분은 HTTP 요청 및 응답에 있는 정보에 매핑됩니다.

`http`

모든 필드는 선택 사항입니다.

- `request` – 요청에 대한 정보입니다.
  - `method` – 요청 메서드입니다. 예: `GET`.
  - `url` – 프로토콜, 호스트 이름과 요청 경로에서 컴파일한 요청의 전체 URL입니다.
  - `user_agent` – 요청자의 클라이언트에 있는 사용자 에이전트 문자열입니다.
  - `client_ip` – 요청자의 IP 주소입니다. IP 패킷의 `Source Address`에서, 또는 전달된 요청인 경우 `X-Forwarded-For` 헤더에서 찾을 수 있습니다.

  - `x_forwarded_for` – (세그먼트 전용) `client_ip`를 `X-Forwarded-For` 헤더에서 읽었으며 위조 염려가 있어 신뢰할 수 없음을 표시하는 부울입니다.
  - `traced` – (하위 세그먼트 전용) 다운스트림 직접 호출이 다른 트레이스된 서비스를 대상으로 함을 표시하는 부울입니다. 이 필드가 `true`로 설정되면, X-Ray는 다운스트림 서비스가 이 블록을 포함한 하위 세그먼트의 `id`와 일치하는 `parent_id`가 있는 세그먼트를 업로드할 때까지는 트레이스가 깨진 것으로 간주합니다.
- `response` – 응답에 대한 정보입니다.
  - `status` – 응답의 HTTP 상태를 표시하는 숫자입니다.
  - `content_length` – 응답 본문의 길이를 바이트로 표시하는 숫자입니다.

다운스트림 웹 API에 대한 직접 호출을 계측할 때 HTTP 요청 및 응답에 대한 정보를 사용하여 하위 세그먼트를 기록합니다. X-Ray는 하위 세그먼트를 사용하여 원격 API에 대해 추정된 세그먼트를 생성합니다.

Example Amazon EC2에서 실행 중인 애플리케이션이 처리하는 HTTP 호출용 세그먼트

```
{
  "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 다운스트림 HTTP 호출에 대한 하위 세그먼트

```
{
  "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 다운스트림 HTTP 호출에 대한 추정된 세그먼트

```
{
  "id": "168416dc2ea97781",
  "name": "names.example.com",
  "trace_id": "1-62be1272-1b71c4274f39f122afa64eab",
  "start_time": 1484786387.131,
  "end_time": 1484786387.501,
  "parent_id": "004f72be19cddc2a",
  "http": {
    "request": {
      "method": "GET",
      "url": "https://names.example.com/"
    },
    "response": {
      "content_length": -1,
      "status": 200
```

```
    }
  },
  "inferred": true
}
```

Annotations

세그먼트와 하위 세그먼트는 X-Ray가 필터 표현식용으로 인덱스하는 필드가 하나 이상 있는 `annotations` 객체를 포함할 수 있습니다. 필드는 문자열, 숫자 또는 부울 값을 가질 수 있습니다(객체 또는 스토리지는 불가). X-Ray가 트레이스당 최대 50개의 주석까지 인덱싱합니다.

Example 주석이 있는 HTTP 호출용 세그먼트

```
{
  "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
    }
  }
```

필터에서 사용하려면 키가 영숫자여야 합니다. 밑줄은 허용됩니다. 다른 기호와 공백은 허용되지 않습니다.

메타데이터

세그먼트와 하위 세그먼트는 객체와 어레이를 포함한 모든 종류의 값을 가진 필드가 하나 이상 있는 `metadata` 객체를 포함할 수 있습니다. X-Ray는 메타데이터를 인덱싱하지 않으며, 세그먼트 문서가 최대 크기(64kB)를 초과하지 않는 이상 값은 크기가 제한되지 않습니다. 메타데이터는 `BatchGetTraces` API가 반환하는 전체 세그먼트 문서에서 확인할 수 있습니다. 로 시작하는 필드 키(debug다음 예제에서는)`AWS.`는 AWS제공 SDK 및 클라이언트용으로 예약되어 SDKs.

Example 메타데이터가 포함된 사용자 지정 하위 세그먼트

```
{
  "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 리소스 데이터

세그먼트의 경우 `aws` 객체에는 애플리케이션을 실행하는 리소스에 대한 정보가 포함됩니다. 여러 필드를 단일 리소스를 적용할 수 있습니다. 예를 들어 Elastic BeanStalk의 멀티컨테이너 Docker 환경에서 실행하는 애플리케이션은 Amazon EC2 인스턴스, 인스턴스를 실행하는 Amazon ECS 컨테이너와 Elastic Beanstalk 환경 자체에 대한 정보를 포함할 수 있습니다.

`aws` (세그먼트)

모든 필드는 선택 사항입니다.

- `account_id` - 애플리케이션이 세그먼트를 다른 로 전송하는 경우 애플리케이션을 실행하는 계정의 ID를 AWS 계정기록합니다.
- `cloudwatch_logs` – 단일 CloudWatch 로그 그룹을 설명하는 객체 배열.
  - `log_group` – CloudWatch 로그 그룹 이름.
  - `arn` – CloudWatch 로그 그룹 ARN.
- `ec2` – EC2 인스턴스에 대한 정보.
  - `instance_id` – EC2 인스턴스의 인스턴스 ID.
  - `instance_size` – EC2 인스턴스 유형.
  - `ami_id` – Amazon Machine Image ID.
  - `availability_zone` – 인스턴스가 실행 중인 가용 영역.
- `ecs` – Amazon ECS 컨테이너 관련 정보.
  - `container` – 컨테이너의 호스트 이름.
  - `container_id` – 컨테이너의 전체 컨테이너 ID.
  - `container_arn` – ECS 컨테이너 인스턴스의 ARN.
- `eks` – Amazon EKS 클러스터에 대한 정보.

  - pod— EKS 포드의 호스트 이름.
  - cluster_name – EKS 클러스터 이름.
  - container_id – 컨테이너의 전체 컨테이너 ID.
- elastic_beanstalk – Elastic Beanstalk 환경에 대한 정보 이 정보는 최신 Elastic Beanstalk 플랫폼의 /var/elasticbeanstalk/xray/environment.conf 파일에서 확인할 수 있습니다.
  - environment_name – 환경의 이름입니다.
  - version_label – 요청을 처리한 인스턴스에 현재 배포된 애플리케이션 버전 이름.
  - deployment_id – 요청을 처리한 인스턴스에 대해 마지막으로 성공한 배포의 ID를 표시하는 숫자.
- xray – 사용된 계측기의 유형 및 버전에 대한 메타데이터.
  - auto_instrumentation – 자동 계측 사용 여부를 나타내는 부울 (예: Java 에이전트).
  - sdk_version – 사용 중인 SDK 또는 에이전트의 버전.
  - sdk – SDK의 유형입니다.

Example AWS 플러그인을 사용한 블록

```
"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"
   }
```

```
}
```

하위 세그먼트의 경우 애플리케이션이 액세스하는 AWS 서비스 및 리소스에 대한 정보를 기록합니다. X-Ray는 이 정보를 사용하여 서비스 맵에 있는 다운스트림 서비스를 나타내는 추론 세그먼트를 생성합니다.

aws (하위 세그먼트)

모든 필드는 선택 사항입니다.

- `operation` - AWS 서비스 또는 리소스에 대해 호출된 API 작업의 이름입니다.
- `account_id` - 애플리케이션이 다른 계정의 리소스에 액세스하거나 세그먼트를 다른 계정으로 전송하는 경우 애플리케이션이 액세스한 AWS 리소스를 소유한 계정의 ID를 기록합니다.
- `region` – 리소스가 애플리케이션과 다른 리전에 있다면, 리전을 기록합니다. 예: `us-west-2`.
- `request_id` – 요청에 대한 고유 식별자입니다.
- `queue_url` – 대기열 작업에서 대기열의 URL을 의미합니다.
- `table_name` – DynamoDB 테이블의 작업에서 테이블의 이름을 의미합니다.

Example 항목을 저장하기 위한 DynamoDB 직접 호출에 대한 하위 세그먼트

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

오류 및 예외

오류가 발생하는 경우, 사용자는 해당 오류와 오류를 일으킨 예외에 대한 정보를 기록할 수 있습니다. 애플리케이션이 사용자에게 오류를 반환하면 세그먼트에, 다운스트림 호출이 오류를 반환하면 하위 세그먼트에 오류를 기록합니다.

오류 유형

다음 필드 중 하나 이상을 `true`로 설정해 오류가 발생했음을 표시합니다. 복합 오류의 경우 여러 유형을 적용할 수 있습니다. 예를 들어 다운스트림 호출에서 발생한 `429 Too Many Requests` 오류는 애플리케이션이 `500 Internal Server Error`를 반환하게 만들기도 하며, 이 경우 3가지 유형이 모두 적용됩니다.

- `error` – 클라이언트 오류가 발생했음을 나타내는 부울입니다(응답 상태 코드가 4XX 클라이언트 오류임).
- `throttle` – 요청이 제한되었음을 나타내는 부울입니다(응답 상태 코드가 429 요청 과다임).
- `fault` – 서버 오류가 발생했음을 나타내는 부울입니다(응답 상태 코드가 5XX 서버 오류임).

세그먼트나 하위 세그먼트에 원인 오브젝트를 포함해 오류의 원인을 표시합니다.

`cause`

원인은 16자 예외 ID나, 다음 필드를 포함한 객체가 될 수 있습니다.

- `working_directory` – 예외 발생 시의 작업 디렉터리 전체 경로입니다.
- `paths` – 예외 발생 시 사용 중인 라이브러리나 모듈의 경로 배열입니다.
- `exceptions` – 예외 객체 배열입니다.

하나 이상의 예외 객체에 있는 오류에 대한 세부 정보를 포함합니다.

`exception`

모든 필드는 선택 사항입니다.

- `id` – 동일한 트레이스 내 세그먼트에서 고유한 예외의 64비트 식별자입니다(16자리 16진수).
- `message` – 예외 메시지입니다.
- `type` – 예외 유형입니다.
- `remote` – 다운스트림 서비스가 반환한 오류 때문에 예외가 발생했음을 나타내는 부울입니다.

- `truncated` – `stack`에서 제외된 스택 숫자를 나타내는 정수입니다.
- `skipped` – 이 예외와 그 하위 항목, 즉 해당 예외가 유발한 예외 간에 건너 뛴 예외 숫자를 표시하는 정수입니다.
- `cause` – 예외의 상위 항목, 즉 해당 예외를 유발한 예외의 ID입니다.
- `stack` – stackFrame 객체 배열입니다.

가능하다면, stackFrame 객체의 호출 스택에 대한 정보를 기록합니다.

`stackFrame`

모든 필드는 선택 사항입니다.

- `path` – 파일의 상대 경로입니다.
- `line` – 파일의 라인입니다.
- `label` – 함수 또는 메서드 이름입니다.

SQL 쿼리

애플리케이션이 SQL 데이터베이스에 만드는 쿼리에 대한 하위 세그먼트를 만들 수 있습니다.

`sql`

모든 필드는 선택 사항입니다.

- `connection_string` – URL 연결 문자열을 사용하지 않는 SQL 서버나 다른 데이터베이스 연결의 경우에는 암호를 제외한 연결 문자열을 기록합니다.
- `url` – URL 연결 문자열을 사용하는 데이터베이스 연결의 경우에는 암호를 제외한 URL을 기록합니다.
- `sanitized_query` – 사용자가 값을 제거하거나 자리 표시자로 대체한 데이터베이스 쿼리입니다.
- `database_type` – 데이터베이스 엔진의 이름입니다.
- `database_version` – 데이터베이스 엔진의 버전 번호입니다.
- `driver_version` – 애플리케이션이 사용하는 데이터베이스 엔진 드라이버의 이름과 버전 번호입니다.
- `user` – 데이터베이스 사용자 이름입니다.
- `preparation` – 쿼리가 `PreparedCall`을 사용한다면 `call`이고, 쿼리가 `PreparedStatement`를 사용한다면 `statement`입니다.

Example SQL 쿼리가 있는 하위 세그먼트

```
{
  "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 개념

AWS X-Ray 는 서비스에서 데이터를 세그먼트로 수신합니다. X-Ray는 공통 요청이 있는 세그먼트를 트레이스로 그룹화합니다. X-Ray는 트레이스를 처리하여 애플리케이션을 시각적으로 표현하는 서비스 그래프를 생성합니다.

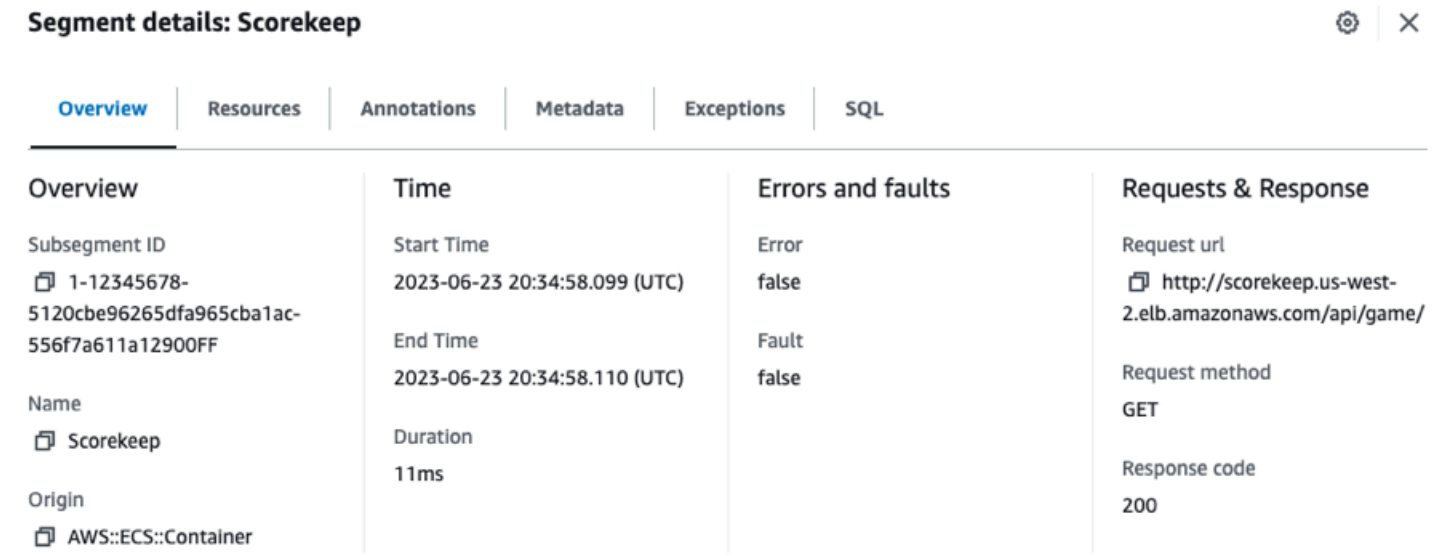
개념

- Segments
- 하위 세그먼트
- 서비스 그래프
- 트레이스
- 샘플링
- 추적 헤더
- 필터 표현식
- Groups
- 주석 및 메타데이터
- 오류, 결함 및 예외

Segments

애플리케이션 로직을 실행하는 컴퓨팅 리소스는 작업에 대한 데이터를 세그먼트의 형식으로 전송합니다. 세그먼트는 리소스의 이름, 요청의 세부 정보, 완료된 작업의 세부 정보를 제공합니다. 예를 들어 HTTP 요청이 애플리케이션에 도달하면 세그먼트가 다음 데이터를 기록할 수 있습니다.

- 호스트 – 호스트 이름, 별칭 또는 IP 주소
- 요청 – 메서드, 클라이언트 주소, 경로, 사용자 에이전트
- 응답 – 상태, 콘텐츠
- 완료된 작업 – 시작 및 종료 시간, 하위 세그먼트
- 발생하는 문제 – 예외 스택의 자동 캡처를 포함한 오류, 결함 및 예외



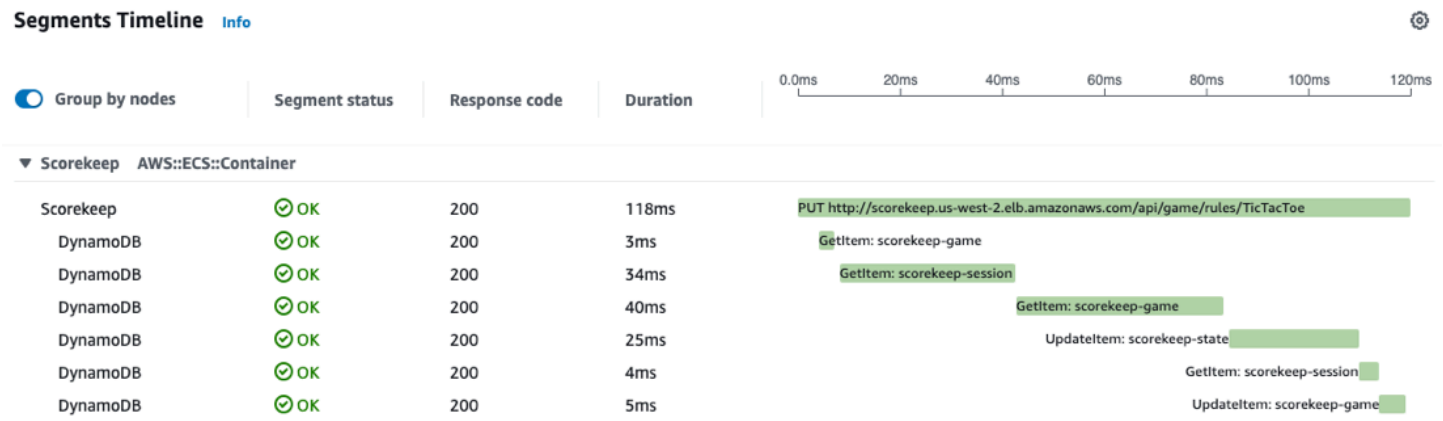
X-Ray SDK는 요청 및 응답 헤더, 애플리케이션의 코드, 애플리케이션이 실행되는 AWS 리소스에 대한 메타데이터에서 정보를 수집합니다. 수신 요청, 다운스트림 요청 및 AWS SDK 클라이언트를 계측하도록 애플리케이션 구성 또는 코드를 수정하여 수집할 데이터를 선택합니다.

> **전달된 요청**
>
> 로드 밸런서 또는 기타 중개자가 애플리케이션으로 요청을 전달하는 경우, X-Ray는 IP 패킷 내 소스 IP가 아니라 요청의 `X-Forwarded-For` 헤더로부터 클라이언트 IP를 가져옵니다. 전달된 요청에 대해 기록된 클라어언트 IP는 위조될 수 있으므로 신뢰하면 안 됩니다.

X-Ray SDK를 사용하여 주석 및 메타데이터와 같은 추가 정보를 레코딩할 수 있습니다. 세그먼트 및 하위 세그먼트에 기록되는 정보 및 구조에 대한 자세한 내용은 AWS X-Ray 세그먼트 문서 단원을 참조하십시오. 세그먼트 문서의 크기는 최대 64kB까지 가능합니다.

하위 세그먼트

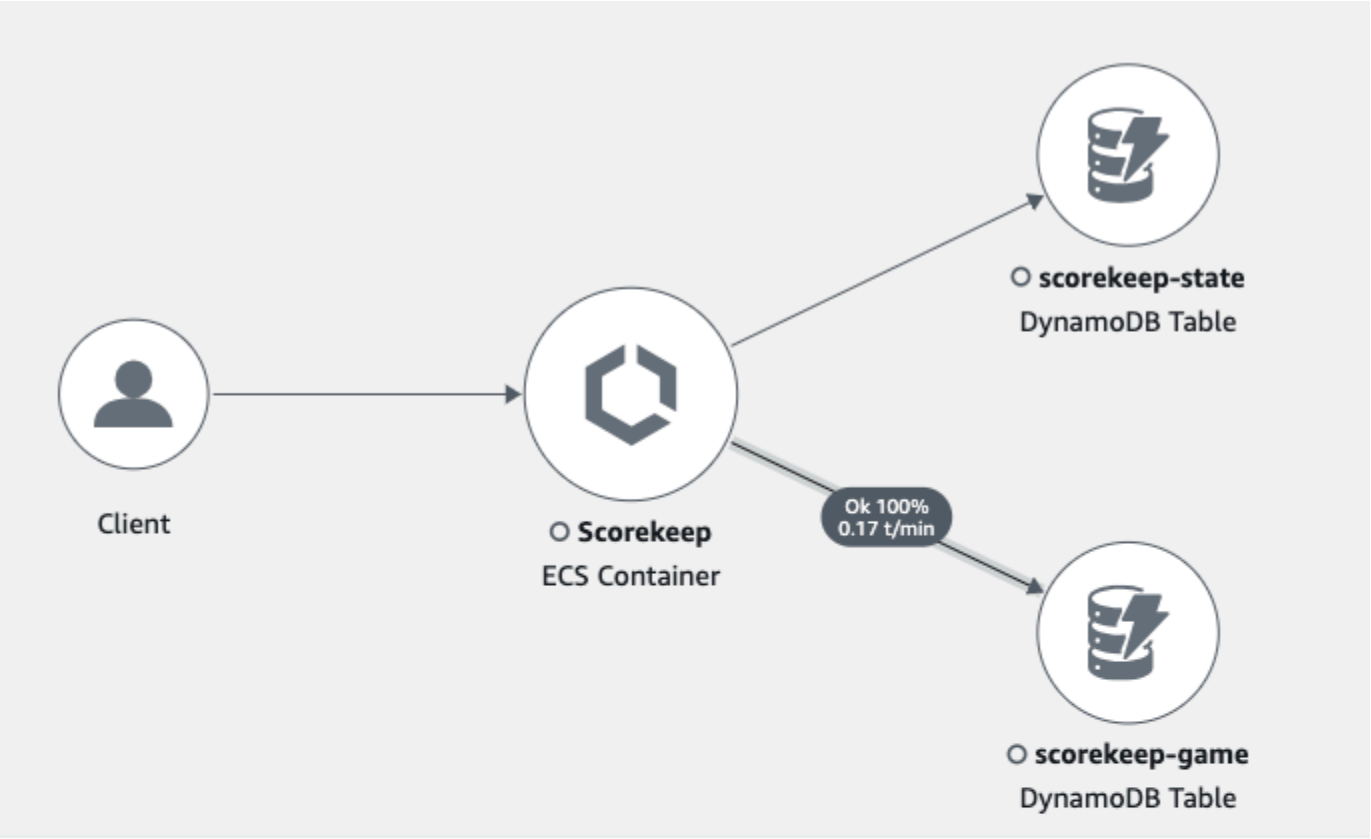
세그먼트는 완료된 작업에 대한 데이터를 하위 세그먼트로 구분할 수 있습니다. 하위 세그먼트는 애플리케이션이 원래 요청을 이행하기 위해 생성한 다운스트림 호출에 대해 보다 세분화된 타이밍 정보 및 세부 정보를 제공합니다. 하위 세그먼트에는 AWS 서비스, 외부 HTTP API 또는 SQL 데이터베이스 호출에 대한 추가 세부 정보가 포함될 수 있습니다. 또한 임의의 하위 세그먼트를 정의하여 애플리케이션의 특정 기능 또는 코드 행을 구성할 수도 있습니다.

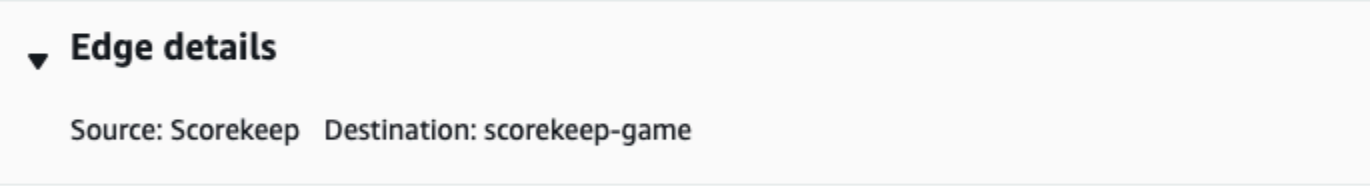


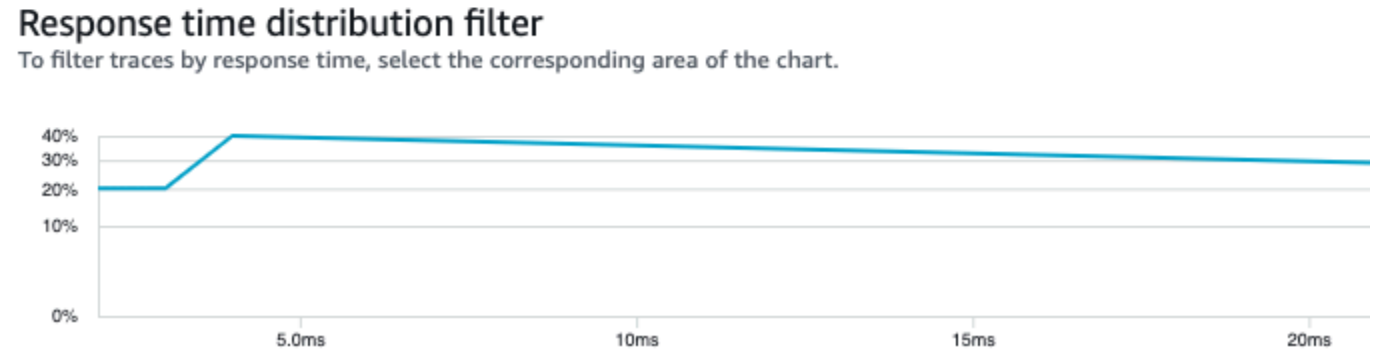
Amazon DynamoDB와 같이 자체 세그먼트를 보내지 않는 서비스의 경우 X-Ray는 하위 세그먼트를 사용하여 트레이스 맵에서 추론된 세그먼트 및 다운스트림 노드를 생성합니다. 이를 통해 다운스트림 종속성이 트레이스를 지원하지 않거나 외부 종속성인 경우에도 해당 종속성을 모두 볼 수 있습니다.

하위 세그먼트는 다운스트림 호출의 애플리케이션 보기를 클라이언트로 표현합니다. 다운스트림 서비스도 구성된 경우 해당 서비스가 보내는 세그먼트는 업스트림 클라이언트의 하위 세그먼트에서 생성되어 추론된 세그먼트를 대체합니다. 서비스 그래프의 노드는 항상 서비스 세그먼트의 정보(사용 가능한 경우)를 사용하며 두 노드 간 엣지는 업스트림 서비스의 하위 세그먼트를 사용합니다.

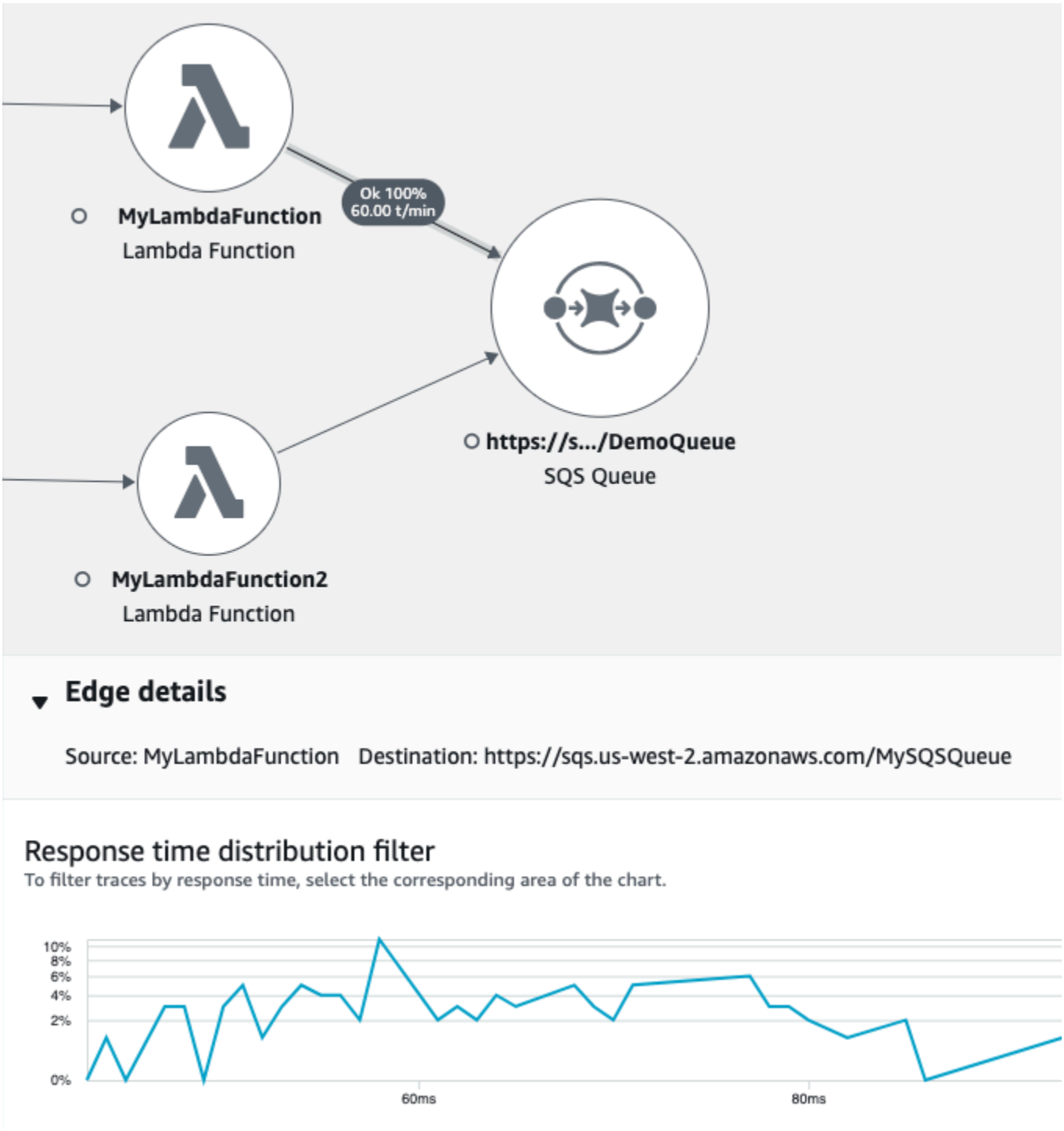
예를 들어 계측된 AWS SDK 클라이언트로 DynamoDB를 호출하면 X-Ray SDK는 해당 호출에 대한 하위 세그먼트를 기록합니다. DynamoDB는 세그먼트를 전송하지 않으므로 트레이스에서 추론된 세그먼트, 서비스 그래프의 DynamoDB 노드, 서비스와 DynamoDB 사이의 엣지에는 모두 하위 세그먼트의 정보가 포함되어 있습니다.







구성된 애플리케이션으로 다른 구성된 서비스를 호출하면 다운스트림 서비스가 해당 세그먼트를 보내 업스트림 서비스가 하위 세그먼트에서 레코딩한 동일한 호출의 해당 보기를 레코딩합니다. 서비스 그래프에서 두 서비스 노드는 해당 서비스 세그먼트의 시간 및 오류 정보를 포함하는 반면 이 둘 간의 엣지는 업스트림 서비스 하위 세그먼트의 정보를 포함합니다.



다운스트림 서비스는 요청에 대한 작업을 시작하고 종료했을 때 정확하게 레코딩하고 업스트림 서비스는 요청이 두 서비스 간을 이동할 때 사용한 시간을 포함한 왕복 지연 시간을 레코딩하므로 두 관점 모두 유용합니다.

서비스 그래프

X-Ray는 애플리케이션이 전송하는 데이터를 사용하여 서비스 그래프를 생성합니다. X-Ray로 데이터를 전송하는 각 AWS 리소스는 그래프에 서비스로 표시됩니다. 엣지는 요청을 처리하기 위해 연동하는 서비스를 연결합니다. 엣지는 클라이언트를 애플리케이션에 연결하고, 애플리케이션을 애플리케이션이 사용하는 다운스트림 서비스 및 리소스에 연결합니다.

> **서비스 이름**
>
> 세그먼트의 name은 해당 세그먼트를 생성한 서비스의 도메인 이름 또는 논리적 이름과 일치해야 합니다. 하지만 이것이 적용되지는 않습니다. `PutTraceSegments` 허가를 받은 애플리케이션은 어떠한 이름을 사용하여도 세그먼트를 전송할 수 있습니다.

서비스 그래프는 애플리케이션을 구성하는 서비스 및 리소스에 대한 정보를 담고 있는 JSON 문서입니다. X-Ray 콘솔은 서비스 그래프를 사용하여 시각화 또는 서비스 맵을 생성합니다.

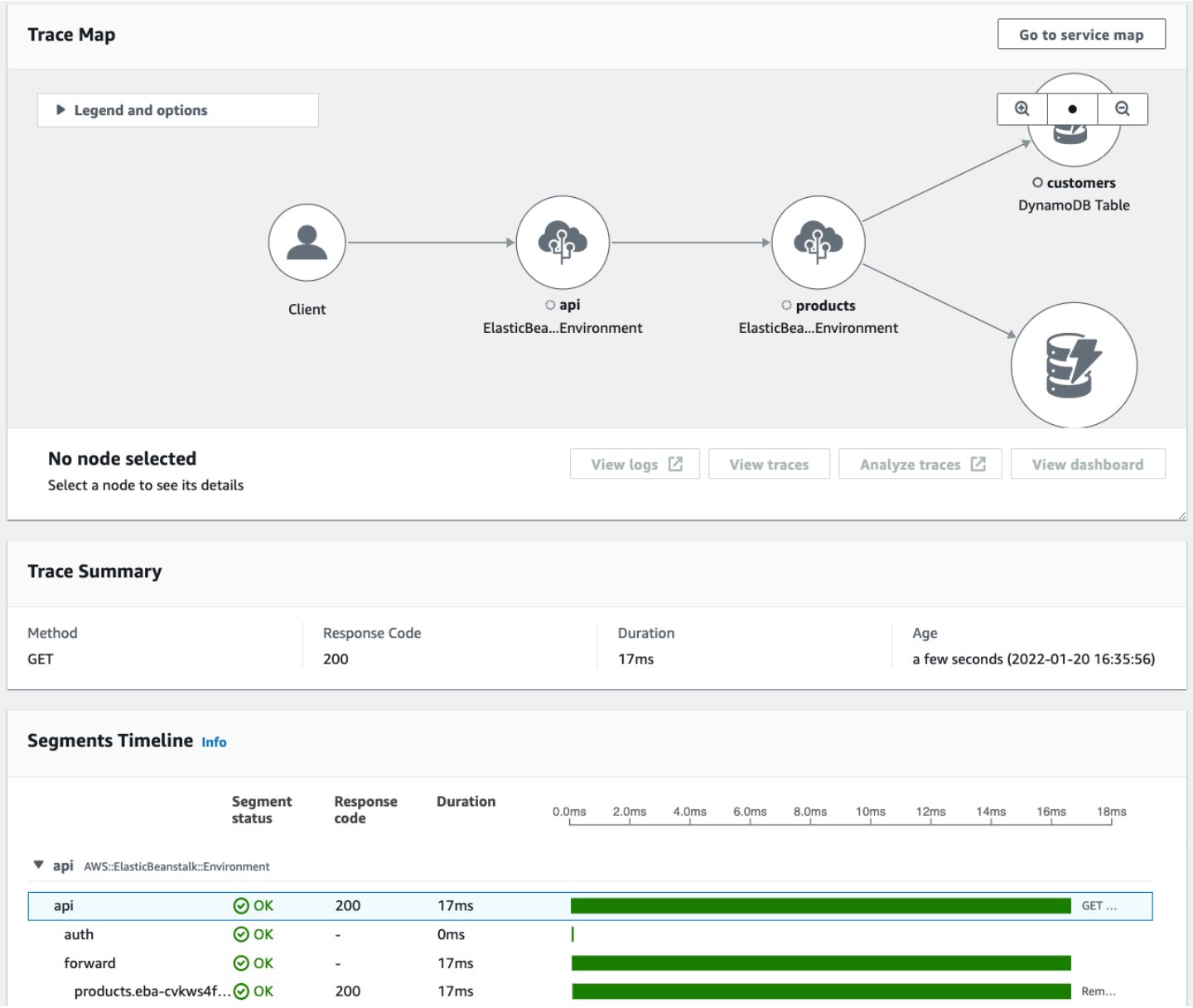
분산된 애플리케이션의 경우, X-Ray가 동일한 트레이스 ID의 요청을 처리하는 모든 서비스의 노드를 하나의 서비스 그래프로 결합합니다. 요청이 도달하는 첫 번째 서비스가 프런트 엔드와 프런트 엔드가 호출하는 서비스 간에 전파되는 트레이스 헤더를 추가합니다.

예를 들어, Scorekeep이 마이크로서비스(AWS Lambda 함수)를 호출하는 웹 API를 실행합니다. 이 마이크로 서비스는 Node.js 라이브러리를 사용하여 임의의 이름을 생성합니다. Java용 X-Ray SDK는 트레이스 ID를 생성하고 이를 Lambda 호출에 포함합니다. Lambda는 트레이스 데이터를 전송하고 트레이스 ID를 함수에 전달합니다. Node.js용 X-Ray SDK도 트레이스 ID를 사용하여 데이터를 전송합니다. 결과적으로 API, Lambda 서비스 및 Lambda 함수에 대한 노드는 모두 트레이스 맵에서 별개의 노드이지만 연결된 노드로 표시됩니다.

서비스 그래프 데이터는 30일 동안 보관됩니다.

트레이스

트레이스 ID는 애플리케이션을 경유하는 요청의 경로를 트레이스합니다. 트레이스는 단일 요청에 의해 생성된 모든 세그먼트를 수집합니다. 이러한 요청은 일반적으로 로드 밸런서를 통과하고, 애플리케이션 코드에 도달하고, 다른 AWS 서비스 또는 외부 웹 API에 대한 다운스트림 호출을 생성하는 HTTP GET 또는 POST 요청입니다. HTTP 요청이 상호 작용하는 첫 번째 지원되는 서비스는 트레이스 ID 헤더를 요청에 추가하고 다운스트림으로 전파하여 지연 시간, 배치 및 기타 요청 데이터를 트레이스합니다.



X-Ray Trace의 청구 방식에 대한 자세한 내용은 AWS X-Ray 요금을 참조하십시오. 트레이스 데이터는 30일 동안 보관됩니다.

샘플링

효율적인 트레이스를 보장하고 애플리케이션에서 처리하는 요청의 대표적 샘플을 제공하기 위해 X-Ray SDK는 샘플링 알고리즘을 적용하여 트레이스되는 요청을 결정합니다. 기본적으로 X-Ray SDK는 매초 첫 번째 요청과 추가 요청의 5%를 기록합니다.

시작할 때 서비스 요금이 발생하지 않도록 하려는 경우 기본 샘플링 비율은 보수적입니다. 기본 샘플링 규칙을 수정하고 서비스 또는 요청의 속성에 따라 샘플링을 적용하는 추가 규칙을 구성하도록 X-Ray를 구성할 수 있습니다.

예를 들어, 샘플링을 비활성화하고 상태를 수정하거나 사용자 또는 트랜잭션을 처리하는 호출에 대한 모든 요청을 트레이스하고자 할 수 있습니다. 백그라운드 폴링, 상태 확인 또는 연결 유지 관리와 같은 대량 읽기 전용 호출 시 낮은 비율로 샘플링하면서도 발생하는 모든 문제를 확인할 수 있을 만큼 충분한 데이터를 확보할 수 있습니다.

자세한 내용은 샘플링 규칙 구성 단원을 참조하십시오.

추적 헤더

모든 요청은 구성 가능한 최소 개수까지 트레이스됩니다. 해당 최소 개수에 도달하면 불필요한 비용을 방지하기 위해 요청의 특정 비율이 트레이스됩니다. 샘플링 결정 및 트레이스 ID가 `X-Amzn-Trace-Id`라는 이름의 트레이스 헤더에서 HTTP 요청에 추가됩니다. 요청이 도달하는 첫 번째 X-Ray 통합 서비스가 트레이스 헤더를 추가합니다. 이 트레이스 헤더는 X-Ray SDK에 의해 판독되며 응답에 포함됩니다.

Example 루트 트레이스 ID 및 샘플링 결정이 포함된 트레이스 헤더

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

> **트레이스 헤더 보안**
>
> 추적 헤더는 X-Ray SDK AWS 서비스, 또는 클라이언트 요청에서 시작될 수 있습니다. 애플리케이션은 수신 요청에서 `X-Amzn-Trace-Id`를 제거하여 사용자가 트레이스 ID 또는 샘플링 결정을 요청에 추가하여 발생하는 문제를 방지할 수 있습니다.

또한 요청이 구성된 애플리케이션에서 생성된 경우 트레이스 헤더에 상위 세그먼트 ID가 포함될 수 있습니다. 예를 들어 애플리케이션이 구성된 HTTP 클라이언트를 사용하여 다운스트림 HTTP 웹 API를 호출하는 경우 X-Ray SDK가 원래 요청에 대한 세그먼트 ID를 다운스트림 요청의 트레이스 헤더에 추가합니다. 다운스트림 요청을 처리하는 구성된 애플리케이션은 상위 세그먼트 ID를 기록하여 두 요청을 연결할 수 있습니다.

Example 트레이스 ID, 상위 세그먼트 ID 및 샘플링 결정이 포함된 트레이스 헤더

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

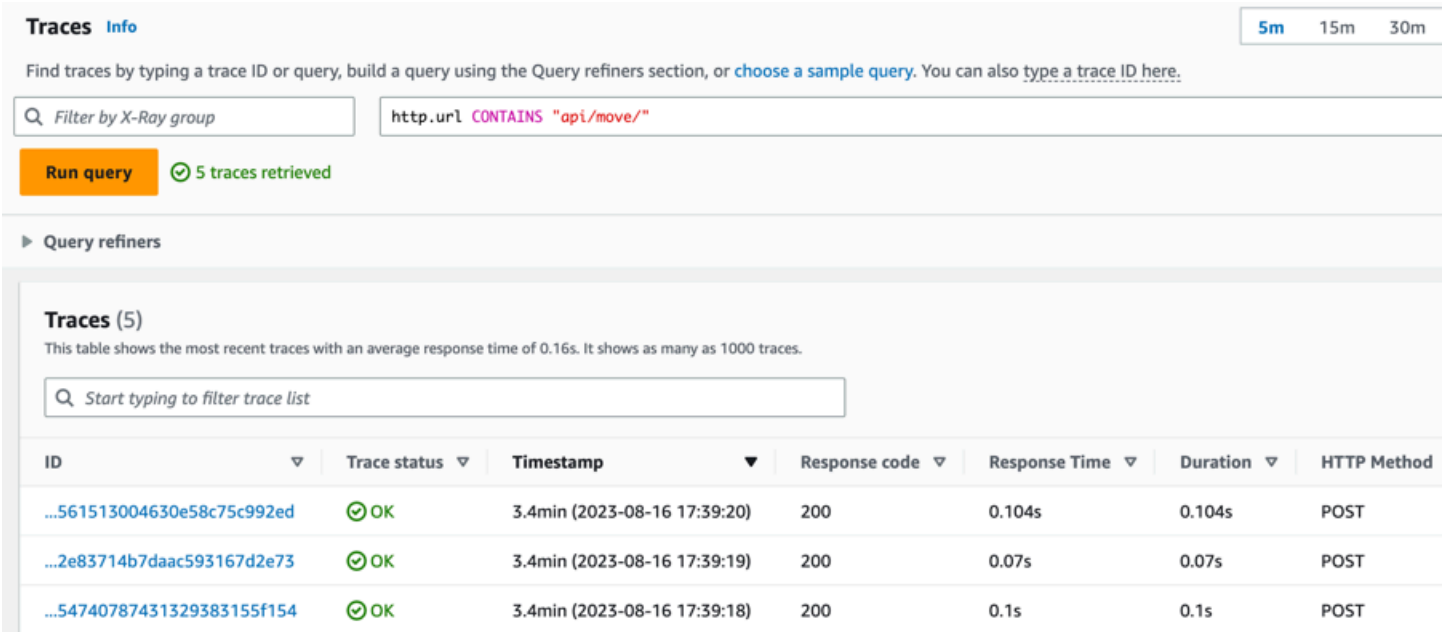
`Lineage`는 Lambda 및 기타에 의해 처리 메커니즘의 AWS 서비스 일부로 트레이스 헤더에 추가될 수 있으며 직접 사용해서는 안 됩니다.

Example 리니지를 이용한 헤더 추적

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

필터 표현식

샘플링을 사용하더라도 복잡한 애플리케이션은 다량의 데이터를 생성합니다. AWS X-Ray 콘솔은 탐색easy-to-navigate 서비스 그래프 보기를 제공합니다. 이 보기는 애플리케이션에서 문제 및 최적화 기회를 쉽게 식별할 수 있는 상태 및 성능 정보를 제공합니다. 고급 트레이스의 경우, 개별 요청에 대해 트레이스를 드릴다운하거나 필터 표현식을 사용하여 특정 경로 또는 사용자와 관련된 트레이스를 찾을 수 있습니다.



Groups

X-Ray는 필터 표현식도 확장하고 그룹 기능도 지원합니다. 필터 표현식을 사용하여 그룹에 추적을 허용하는 기준을 정의할 수 있습니다.

이름 또는 Amazon 리소스 이름(ARN)으로 그룹을 직접 호출하여 자체 서비스 그래프, 추적 요약 및 Amazon CloudWatch 지표를 생성할 수 있습니다. 그룹이 생성되면 수신 추적은 X-Ray 서비스에 저장될 때 그룹의 필터 표현식에 대해 확인됩니다. 각 기준과 일치하는 추적 수에 대한 지표는 매분마다 CloudWatch에 게시됩니다.

그룹의 필터 표현식을 업데이트해도 이미 기록한 데이터는 변경되지 않습니다. 업데이트는 후속 추적에만 적용됩니다. 이렇게 하면 새 표현식과 이전 표현식의 그래프를 병합할 수 있습니다. 이를 방지하려면 현재 그룹을 삭제하고 새로 만드십시오.

> **Note**
>
> 그룹은 필터 표현식과 일치하는 검색 완료 트레이스의 수로 청구됩니다. 자세한 내용은 AWS X-Ray 요금을 참조하세요.

그룹에 대한 자세한 정보는 그룹 구성 섹션을 참조하세요.

주석 및 메타데이터

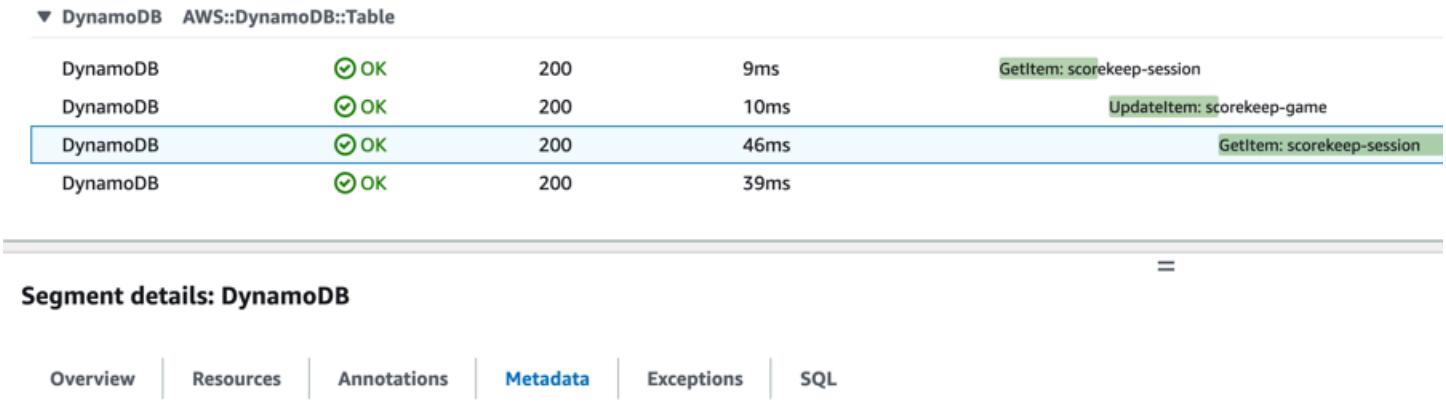
애플리케이션을 계측할 때 X-Ray SDK는 수신 및 발신 요청, 사용된 AWS 리소스 및 애플리케이션 자체에 대한 정보를 기록합니다. 세그먼트 문서에 주석 및 메타데이터로 다른 정보를 추가할 수 있습니다. 주석 및 메타데이터는 트레이스 수준에서 집계되며 세그먼트 또는 하위 세그먼트에 추가할 수 있습니다.

주석은 필터 표현식에 사용하도록 인덱싱된 단순한 키-값 페어입니다. 주석은 콘솔의 트레이스를 그룹화할 때 사용할 데이터를 기록하거나 `GetTraceSummaries` API를 직접 호출할 때 사용하세요.

X-Ray가 트레이스당 최대 50개의 주석까지 인덱싱합니다.

메타데이터는 객체와 목록을 포함한 모든 유형의 키-값 페어이지만 인덱싱되어 있지는 않습니다. 메타데이터를 사용하여 트레이스에 저장하고 싶지만 트레이스 검색에는 사용하지 않을 데이터를 기록합니다.

CloudWatch 콘솔의 트레이스 세부 정보 페이지에 있는 세그먼트 또는 하위 세그먼트 세부 정보에서 주석 및 메타데이터를 볼 수 있습니다.



오류, 결함 및 예외

X-Ray는 애플리케이션 코드에서 발생하는 오류 및 다운스트림 서비스가 반환하는 오류를 추적합니다. 오류는 다음과 같이 분류됩니다.

- **`Error`** – 클라이언트 오류(400 시리즈 오류)
- **`Fault`** – 서버 장애(500 시리즈 오류)
- **`Throttle`** – 제한 오류(429 요청 과다)

애플리케이션이 구성된 요청을 처리하는 동안 예외가 발생하면 X-Ray SDK가 스택 트레이스(사용 가능한 경우)를 포함하여 예외에 대한 세부 정보를 레코딩합니다. X-Ray 콘솔의 세그먼트 세부 정보에서 예외를 볼 수 있습니다.

의 보안 AWS X-Ray

의 클라우드 보안 AWS 이 최우선 순위입니다. AWS 고객은 보안에 가장 민감한 조직의 요구 사항을 충족하도록 구축된 데이터 센터 및 네트워크 아키텍처의 이점을 누릴 수 있습니다.

보안은 AWS 와 사용자 간의 공동 책임입니다. 공동 책임 모델은 이 사항을 클라우드 내 보안 및 클라우드의 보안으로 설명합니다.

- 클라우드 보안 - AWS AWS 서비스 에서 실행되는 인프라를 보호할 책임이 있습니다 AWS 클라우드. AWS 또한는 안전하게 사용할 수 있는 서비스를 제공합니다. 서드 파티 감사자는 정기적으로 AWS 규정 준수 프로그램의 일환으로 보안 효과를 테스트하고 검증합니다. X-Ray에 적용되는 규정 준수 프로그램에 대한 자세한 내용은 규정 준수 프로그램 제공 범위 내의AWS 서비스를 참조하세요.
- 클라우드의 보안 - 사용자의 책임은 AWS 서비스 사용하는에 따라 결정됩니다. 또한 데이터의 민감도, 조직의 요건 및 관련 법률 및 규정을 비롯한 기타 요소에 대해서도 책임이 있습니다.

이 설명서는 X-Ray 사용 시 공동 책임 모델을 적용하는 방법을 이해하는 데 도움이 됩니다. 다음 주제에서는 보안 및 규정 준수 목표를 충족하도록 X-Ray를 구성하는 방법을 보여줍니다. 또한 X-Ray 리소스를 모니터링하고 보호하는 데 도움이 AWS 서비스 되는 다른를 사용하는 방법을 알아봅니다.

주제

- 의 데이터 보호 AWS X-Ray
- 에 대한 자격 증명 및 액세스 관리 AWS X-Ray
- 에 대한 규정 준수 검증 AWS X-Ray
- 의 복원성 AWS X-Ray
- 의 인프라 보안 AWS X-Ray

의 데이터 보호 AWS X-Ray

AWS X-Ray 는 항상 추적 및 관련 저장 데이터를 암호화합니다. 규정 준수 또는 내부 요구 사항에 대해 암호화 키를 감사하고 비활성화해야 하는 경우 AWS Key Management Service (AWS KMS) 키를 사용하여 데이터를 암호화하도록 X-Ray를 구성할 수 있습니다.

X-Ray는 AWS 관리형 키 이름이 인를 제공합니다`aws/xray`. AWS CloudTrail에서 키 사용을 감사하고 키 자체를 관리할 필요가 없으면 이 키를 사용하십시오. 키에 대한 액세스를 관리하거나 키 교체를 구성해야 할 경우 고객 관리 키를 생성할 수 있습니다.

암호화 설정을 변경하면 X-Ray가 데이터 키를 생성하고 전파하느라 시간이 다소 소모됩니다. 새 키를 처리하는 동안 X-Ray가 새 설정과 기존 설정을 조합하여 데이터를 암호화할 수 있습니다. 암호화 설정을 변경할 때 기존 데이터는 재암호화하지 않습니다.

> **Note**
>
> AWS KMS X-Ray가 KMS 키를 사용하여 추적 데이터를 암호화하거나 해독하는 경우에 요금이 부과됩니다.
>
> - 기본 암호화 – 무료.
> - AWS 관리형 키— 키 사용료를 지불하세요.
> - 고객 관리 키 – 키 보관 및 사용료가 부과됩니다.
>
> 자세한 내용은 AWS Key Management Service 요금을 참조하세요.

> **Note**
>
> X-Ray 인사이트 알림은 현재 고객 관리 키를 지원하지 않는 Amazon EventBridge로 이벤트를 전송합니다. 자세한 내용은 Amazon EventBridge의 데이터 보호를 참조하십시오.

고객 관리 키를 사용하여 암호화된 트레이스를 보도록 X-Ray를 구성하려면 고객 관리 키에 대한 사용자 수준 액세스 권한이 있어야 합니다. 자세한 내용은 암호화에 대한 사용자 권한 섹션을 참조하세요.

CloudWatch console

CloudWatch 콘솔을 사용하여 암호화에 KMS 키를 사용하도록 X-Ray를 구성하려면

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/cloudwatch/:// https://https://://https://://https://://CloudWatch://://https://://https://://https://https://://https://:// https
2. 왼쪽 탐색 창에서 설정을 선택합니다.
3. X-Ray 추적 섹션의 암호화에서 설정 보기를 선택합니다.
4. 암호화 구성 섹션에서 편집을 선택합니다.
5. KMS 키 사용을 선택합니다.
6. 드롭다운 메뉴에서 키를 선택합니다.

- aws/xray — AWS 관리형 키를 사용하십시오.
- 키 별칭 – 계정에서 고객 관리형 CMK를 사용합니다.
- 키 ARN 수동 입력 – 다른 계정에 있는 고객 관리 키를 사용합니다. 나타나는 필드에 키의 Amazon 리소스 이름(ARN)을 전체 다 입력합니다.

7. Update encryption (암호화 업데이트)를 선택합니다.

X-Ray console

X-Ray 콘솔을 사용하여 암호화에 KMS 키를 사용하도록 X-Ray를 구성하려면

1. X-Ray 콘솔을 엽니다.
2. 암호화를 선택합니다.
3. KMS 키 사용을 선택합니다.
4. 드롭다운 메뉴에서 키를 선택합니다.

- aws/xray — AWS 관리형 키를 사용하십시오.
- 키 별칭 – 계정에서 고객 관리형 CMK를 사용합니다.
- 키 ARN 수동 입력 – 다른 계정에 있는 고객 관리 키를 사용합니다. 나타나는 필드에 키의 Amazon 리소스 이름(ARN)을 전체 다 입력합니다.

5. Apply(적용)를 선택합니다.

> **Note**
>
> X-Ray는 비대칭 KMS 키를 지원하지 않습니다.

X-Ray가 암호화 키에 액세스하지 못하면 데이터 저장이 중단됩니다. KMS 키가 액세스를 잃어버리거나 현재 사용 중인 키를 비활성화하는 경우 이런 일이 발생합니다. 이 경우 X-Ray가 탐색 창에 알림을 표시합니다.

X-Ray API로 암호화 설정을 구성하려면 AWS X-Ray API로 샘플링, 그룹 및 암호화 설정 구성 섹션을 참조하십시오.

에 대한 자격 증명 및 액세스 관리 AWS X-Ray

AWS Identity and Access Management (IAM)는 관리자가 AWS 리소스에 대한 액세스를 안전하게 제어하는 데 도움이 AWS 서비스 되는 입니다. IAM 관리자는 누가 X-Ray 리소스를 사용하도록 인증되고(로그인됨) 권한이 부여되는지(권한 있음)를 제어합니다. IAM은 추가 비용 없이 사용할 수 AWS 서비스 있는 입니다.

주제

대상

AWS Identity and Access Management (IAM)를 사용하는 방법은 X-Ray에서 수행하는 작업에 따라 다릅니다.

서비스 사용자 – X-Ray 서비스를 사용하여 작업을 수행하는 경우 필요한 자격 증명과 권한을 관리자가 제공합니다. 더 많은 X-Ray 기능을 사용하여 작업을 수행하게 되면 추가 권한이 필요할 수 있습니다. 액세스 권한 관리 방법을 이해하면 관리자에게 올바른 권한을 요청하는 데 도움이 됩니다. X-Ray의 기능에 액세스할 수 없는 경우 자격 AWS X-Ray 증명 및 액세스 문제 해결 섹션을 참조하세요.

서비스 관리자 - 회사에서 X-Ray 리소스를 책임지고 있는 경우 X-Ray에 대한 전체 액세스 권한을 가지고 있을 것입니다. 서비스 관리자는 서비스 사용자가 액세스해야 하는 X-Ray 기능과 리소스를 결정합니다. 그런 다음 IAM 관리자에게 요청을 제출하여 서비스 사용자의 권한을 변경해야 합니다. 이 페이지의 정보를 검토하여 IAM의 기본 개념을 이해하세요. 회사가 X-Ray에서 IAM을 사용하는 방법에 대해 자세히 알아보려면 AWS X-Ray 에서 IAM을 사용하는 방법 섹션을 참조하세요.

IAM 관리자 - IAM 관리자라면 X-Ray에 대한 액세스 권한 관리 정책 작성 방법을 자세히 알고 싶을 것입니다. IAM에서 사용할 수 있는 X-Ray 자격 증명 기반 정책 예제를 보려면 AWS X-Ray 자격 증명 기반 정책 예제 섹션을 참조하세요.

ID를 통한 인증

인증은 자격 증명 자격 증명을 AWS 사용하여에 로그인하는 방법입니다. IAM 사용자 또는 AWS 계정 루트 사용자 IAM 역할을 수임하여 로 인증(로그인 AWS)되어야 합니다.

자격 증명 소스를 통해 제공된 자격 증명을 사용하여 페더레이션 자격 증명 AWS 으로에 로그인할 수 있습니다. AWS IAM Identity Center (IAM Identity Center) 사용자, 회사의 Single Sign-On 인증 및 Google 또는 Facebook 자격 증명은 페더레이션 자격 증명의 예입니다. 페더레이션형 ID로 로그인할 때 관리자가 이전에 IAM 역할을 사용하여 ID 페더레이션을 설정했습니다. 페더레이션을 사용하여 AWS 에 액세스하면 간접적으로 역할을 수임하게 됩니다.

사용자 유형에 따라 AWS Management Console 또는 AWS 액세스 포털에 로그인할 수 있습니다. 로그인에 대한 자세한 내용은 AWS 로그인 사용 설명서의에 로그인하는 방법을 AWS참조하세요. AWS 계정

AWS 프로그래밍 방식으로에 액세스하는 경우는 자격 증명을 사용하여 요청에 암호화 방식으로 서명할 수 있는 소프트웨어 개발 키트(SDK)와 명령줄 인터페이스(CLI)를 AWS 제공합니다. AWS 도구를 사용하지 않는 경우 요청에 직접 서명해야 합니다. 권장 방법을 사용하여 요청에 직접 서명하는 자세한 방법은 IAM 사용 설명서에서 API 요청용AWS Signature Version 4를 참조하세요.

사용하는 인증 방법에 상관없이 추가 보안 정보를 제공해야 할 수도 있습니다. 예를 들어, 다중 인증(MFA)을 사용하여 계정의 보안을 강화하는 것이 AWS 좋습니다. 자세한 내용은 AWS IAM Identity Center 사용 설명서에서 다중 인증 및 IAM 사용 설명서에서 IAM의AWS 다중 인증을 참조하세요.

AWS 계정 루트 사용자

를 생성할 때 계정의 모든 AWS 서비스 및 리소스에 대한 완전한 액세스 권한이 있는 하나의 로그인 자격 증명으로 AWS 계정시작합니다. 이 자격 증명을 AWS 계정 루트 사용자라고 하며 계정을 생성하는 데 사용한 이메일 주소와 암호로 로그인하여 액세스합니다. 일상적인 작업에 루트 사용자를 사용하지 않을 것을 강력히 권장합니다. 루트 사용자 자격 증명을 보호하고 루트 사용자만 수행할 수 있는 작업을 수행하는 데 사용합니다. 루트 사용자로 로그인해야 하는 전체 작업 목록은 IAM 사용 설명서의 루트 사용자 자격 증명이 필요한 작업을 참조하세요.

IAM 사용자 및 그룹

IAM 사용자는 단일 사용자 또는 애플리케이션에 대한 특정 권한이 AWS 계정 있는 내의 자격 증명입니다. 가능하면 암호 및 액세스 키와 같은 장기 자격 증명이 있는 IAM 사용자를 생성하는 대신 임시 자격 증명을 사용하는 것이 좋습니다. 하지만 IAM 사용자의 장기 자격 증명이 필요한 특정 사용 사례가 있는 경우, 액세스 키를 교체하는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서의 장기 보안 인증이 필요한 사용 사례의 경우, 정기적으로 액세스 키 교체를 참조하세요.

IAM 그룹은 IAM 사용자 컬렉션을 지정하는 자격 증명입니다. 사용자는 그룹으로 로그인할 수 없습니다. 그룹을 사용하여 여러 사용자의 권한을 한 번에 지정할 수 있습니다. 그룹을 사용하면 대규모 사용자 집합의 권한을 더 쉽게 관리할 수 있습니다. 예를 들어, IAMAdmins라는 그룹이 있고 이 그룹에 IAM 리소스를 관리할 권한을 부여할 수 있습니다.

사용자는 역할과 다릅니다. 사용자는 한 사람 또는 애플리케이션과 고유하게 연결되지만, 역할은 해당 역할이 필요한 사람이라면 누구나 수임할 수 있습니다. 사용자는 영구적인 장기 자격 증명을 가지고 있지만, 역할은 임시 보안 인증만 제공합니다. 자세한 내용은 IAM 사용 설명서에서 IAM 사용자 사용 사례를 참조하세요.

IAM 역할

IAM 역할은 특정 권한이 AWS 계정 있는 내의 자격 증명입니다. IAM 사용자와 유사하지만, 특정 개인과 연결되지 않습니다. 에서 IAM 역할을 일시적으로 수임하려면 사용자에서 IAM 역할(콘솔)로 전환할 AWS Management Console수 있습니다. https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_use_switch-role-console.html 또는 AWS API 작업을 호출하거나 사용자 지정 URL을 AWS CLI 사용하여 역할을 수임할 수 있습니다. 역할 사용 방법에 대한 자세한 내용은 IAM 사용 설명서의 역할 수임 방법을 참조하세요.

임시 보안 인증이 있는 IAM 역할은 다음과 같은 상황에서 유용합니다.

- 페더레이션 사용자 액세스 - 페더레이션 ID에 권한을 부여하려면 역할을 생성하고 해당 역할의 권한을 정의합니다. 페더레이션 ID가 인증되면 역할이 연결되고 역할에 정의된 권한이 부여됩니다. 페더레이션 관련 역할에 대한 자세한 내용은 IAM 사용 설명서의 Create a role for a third-party identity provider (federation)를 참조하세요. IAM Identity Center를 사용하는 경우, 권한 집합을 구성합니다. 인증 후 ID가 액세스할 수 있는 항목을 제어하기 위해 IAM Identity Center는 권한 집합을 IAM의 역할과 연관짓습니다. 권한 집합에 대한 자세한 내용은 AWS IAM Identity Center 사용 설명서의 권한 집합을 참조하세요.
- 임시 IAM 사용자 권한 - IAM 사용자 또는 역할은 IAM 역할을 수임하여 특정 작업에 대한 다양한 권한을 임시로 받을 수 있습니다.
- 교차 계정 액세스 - IAM 역할을 사용하여 다른 계정의 사용자(신뢰할 수 있는 보안 주체)가 내 계정의 리소스에 액세스하도록 허용할 수 있습니다. 역할은 계정 간 액세스를 부여하는 기본적인 방법입니다. 그러나 일부 에서는 정책을 리소스에 직접 연결할 AWS 서비스수 있습니다(역할을 프록시로 사용하는 대신). 교차 계정 액세스에 대한 역할과 리소스 기반 정책의 차이점을 알아보려면 IAM 사용 설명서의 IAM의 교차 계정 리소스 액세스를 참조하세요.
- 교차 서비스 액세스 - 일부는 다른에서 기능을 AWS 서비스 사용합니다 AWS 서비스. 예를 들어, 서비스에서 호출하면 일반적으로 해당 서비스는 Amazon EC2에서 애플리케이션을 실행하거나

Amazon S3에 객체를 저장합니다. 서비스는 직접적으로 호출하는 위탁자의 권한을 사용하거나, 서비스 역할을 사용하거나, 또는 서비스 연결 역할을 사용하여 이 작업을 수행할 수 있습니다.

- 전달 액세스 세션(FAS) - IAM 사용자 또는 역할을 사용하여에서 작업을 수행하는 경우 AWS보안 주체로 간주됩니다. 일부 서비스를 사용하는 경우, 다른 서비스에서 다른 작업을 시작하는 작업을 수행할 수 있습니다. FAS는를 호출하는 보안 주체의 권한을 다운스트림 서비스에 AWS 서비스 대한 요청과 AWS 서비스함께 사용합니다. FAS 요청은 서비스가 완료하기 위해 다른 AWS 서비스 또는 리소스와의 상호 작용이 필요한 요청을 수신할 때만 이루어집니다. 이 경우, 두 작업을 모두 수행할 수 있는 권한이 있어야 합니다. FAS 요청 시 정책 세부 정보는 전달 액세스 세션을 참조하세요.
- 서비스 역할 - 서비스 역할은 서비스가 사용자를 대신하여 작업을 수행하기 위해 맡는 IAM 역할입니다. IAM 관리자는 IAM 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다. 자세한 정보는 IAM 사용 설명서의 Create a role to delegate permissions to an AWS 서비스를 참조하세요.
- 서비스 연결 역할 - 서비스 연결 역할은에 연결된 서비스 역할의 한 유형입니다 AWS 서비스. 서비스는 사용자를 대신하여 작업을 수행하기 위해 역할을 수임할 수 있습니다. 서비스 연결 역할은에 나타나 AWS 계정 며 서비스가 소유합니다. IAM 관리자는 서비스 링크 역할의 권한을 볼 수 있지만 편집은 할 수 없습니다.

- Amazon EC2에서 실행되는 애플리케이션 - IAM 역할을 사용하여 EC2 인스턴스에서 실행되고 AWS CLI 또는 AWS API 요청을 수행하는 애플리케이션의 임시 자격 증명을 관리할 수 있습니다. 이는 EC2 인스턴스 내에 액세스 키를 저장할 때 권장되는 방법입니다. EC2 인스턴스에 AWS 역할을 할당하고 모든 애플리케이션에서 사용할 수 있도록 하려면 인스턴스에 연결된 인스턴스 프로파일을 생성합니다. 인스턴스 프로필에는 역할이 포함되어 있으며 EC2 인스턴스에서 실행되는 프로그램이 임시 보안 인증을 얻을 수 있습니다. 자세한 정보는 IAM 사용 설명서의 IAM 역할을 사용하여 Amazon EC2 인스턴스에서 실행되는 애플리케이션에 권한 부여를 참조하세요.

정책을 사용하여 액세스 관리

정책을 AWS 생성하고 자격 증명 또는 리소스에 연결하여 AWS 에서 액세스를 제어합니다. 정책은 자격 증명 또는 리소스와 연결된 AWS 경우 권한을 정의하는의 객체입니다.는 보안 주체(사용자, 루트 사용자 또는 역할 세션)가 요청할 때 이러한 정책을 AWS 평가합니다. 정책에서 권한은 요청이 허용되거나 거부되는 지를 결정합니다. 대부분의 정책은에 JSON 문서 AWS 로 저장됩니다. JSON 정책 문서의 구조와 콘텐츠에 대한 자세한 정보는 IAM 사용 설명서의 JSON 정책 개요를 참조하세요.

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

기본적으로, 사용자 및 역할에는 어떠한 권한도 없습니다. 사용자에게 사용자가 필요한 리소스에서 작업을 수행할 권한을 부여하려면 IAM 관리자가 IAM 정책을 생성하면 됩니다. 그런 다음 관리자가 IAM 정책을 역할에 추가하고, 사용자가 역할을 수임할 수 있습니다.

IAM 정책은 작업을 수행하기 위해 사용하는 방법과 상관없이 작업에 대한 권한을 정의합니다. 예를 들어, `iam:GetRole` 작업을 허용하는 정책이 있다고 가정합니다. 해당 정책이 있는 사용자는 AWS Management Console AWS CLI, 또는 API에서 역할 정보를 가져올 수 있습니다 AWS .

ID 기반 정책

ID 기반 정책은 IAM 사용자, 사용자 그룹 또는 역할과 같은 ID에 연결할 수 있는 JSON 권한 정책 문서입니다. 이러한 정책은 사용자 및 역할이 어떤 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 제어합니다. 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서에서 고객 관리형 정책으로 사용자 지정 IAM 권한 정의를 참조하세요.

ID 기반 정책은 인라인 정책 또는 관리형 정책으로 한층 더 분류할 수 있습니다. 인라인 정책은 단일 사용자, 그룹 또는 역할에 직접 포함됩니다. 관리형 정책은의 여러 사용자, 그룹 및 역할에 연결할 수 있는 독립 실행형 정책입니다 AWS 계정. 관리형 정책에는 AWS 관리형 정책 및 고객 관리형 정책이 포함됩니다. 관리형 정책 또는 인라인 정책을 선택하는 방법을 알아보려면 IAM 사용 설명서의 관리형 정책 및 인라인 정책 중에서 선택을 참조하세요.

리소스 기반 정책

리소스 기반 정책은 리소스에 연결하는 JSON 정책 설명서입니다. 리소스 기반 정책의 예제는 IAM 역할 신뢰 정책과 Amazon S3 버킷 정책입니다. 리소스 기반 정책을 지원하는 서비스에서 서비스 관리자는 이러한 정책을 사용하여 특정 리소스에 대한 액세스를 통제할 수 있습니다. 정책이 연결된 리소스의 경우 정책은 지정된 위탁자가 해당 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 정의합니다. 리소스 기반 정책에서 위탁자를 지정해야 합니다. 보안 주체에는 계정, 사용자, 역할, 페더레이션 사용자 또는이 포함될 수 있습니다 AWS 서비스.

리소스 기반 정책은 해당 서비스에 있는 인라인 정책입니다. 리소스 기반 정책에서는 IAM의 AWS 관리형 정책을 사용할 수 없습니다.

액세스 제어 목록(ACL)

액세스 제어 목록(ACL)은 어떤 보안 주체(계정 멤버, 사용자 또는 역할)가 리소스에 액세스할 수 있는 권한을 가지고 있는지를 제어합니다. ACL은 JSON 정책 문서 형식을 사용하지 않지만 리소스 기반 정책과 유사합니다.

Amazon S3 AWS WAF및 Amazon VPC는 ACLs. ACL에 관한 자세한 내용은 Amazon Simple Storage Service 개발자 가이드의 액세스 제어 목록(ACL) 개요를 참조하세요.

기타 정책 타입

AWS 는 덜 일반적인 추가 정책 유형을 지원합니다. 이러한 정책 타입은 더 일반적인 정책 유형에 따라 사용자에게 부여되는 최대 권한을 설정할 수 있습니다.

- 권한 경계 – 권한 경계는 ID 기반 정책에 따라 IAM 엔터티(IAM 사용자 또는 역할)에 부여할 수 있는 최대 권한을 설정하는 고급 기능입니다. 개체에 대한 권한 경계를 설정할 수 있습니다. 그 결과로 얻는 권한은 객체의 ID 기반 정책과 그 권한 경계의 교집합입니다. `Principal` 필드에서 사용자나 역할을 지정하는 리소스 기반 정책은 권한 경계를 통해 제한되지 않습니다. 이러한 정책 중 하나에 포함된 명시적 거부는 허용을 재정의합니다. 권한 경계에 대한 자세한 정보는 IAM 사용 설명서의 IAM 엔티티에 대한 권한 경계를 참조하세요.
- 서비스 제어 정책(SCPs) - SCPs는의 조직 또는 조직 단위(OU)에 대한 최대 권한을 지정하는 JSON 정책입니다 AWS Organizations. AWS Organizations 는 비즈니스가 소유 AWS 계정 한 여러를 그룹화하고 중앙에서 관리하기 위한 서비스입니다. 조직에서 모든 기능을 활성화할 경우, 서비스 제어 정책(SCP)을 임의의 또는 모든 계정에 적용할 수 있습니다. SCP는 각각을 포함하여 멤버 계정의 엔터티에 대한 권한을 제한합니다 AWS 계정 루트 사용자. 조직 및 SCP에 대한 자세한 내용은 AWS Organizations 사용 설명서에서 Service control policies을 참조하세요.
- 리소스 제어 정책(RCP) - RCP는 소유한 각 리소스에 연결된 IAM 정책을 업데이트하지 않고 계정의 리소스에 대해 사용 가능한 최대 권한을 설정하는 데 사용할 수 있는 JSON 정책입니다. RCP는 멤버 계정의 리소스에 대한 권한을 제한하며 조직에 속하는지 여부에 AWS 계정 루트 사용자관계없이를 포함한 자격 증명에 대한 유효 권한에 영향을 미칠 수 있습니다. RCP를 AWS 서비스 지원하는 목록을 포함하여 조직 및 RCPs에 대한 자세한 내용은 AWS Organizations 사용 설명서의 리소스 제어 정책(RCPs)을 참조하세요.
- 세션 정책 – 세션 정책은 역할 또는 페더레이션 사용자에 대해 임시 세션을 프로그래밍 방식으로 생성할 때 파라미터로 전달하는 고급 정책입니다. 결과적으로 얻는 세션의 권한은 사용자 또는 역할의 ID 기반 정책의 교차와 세션 정책입니다. 또한 권한을 리소스 기반 정책에서 가져올 수도 있습니다. 이러한 정책 중 하나에 포함된 명시적 거부는 허용을 재정의합니다. 자세한 정보는 IAM 사용 설명서의 세션 정책을 참조하세요.

여러 정책 유형

여러 정책 유형이 요청에 적용되는 경우, 결과 권한은 이해하기가 더 복잡합니다. 에서 여러 정책 유형이 관련될 때 요청을 허용할지 여부를 AWS 결정하는 방법을 알아보려면 IAM 사용 설명서의 정책 평가 로직을 참조하세요.

AWS X-Ray 에서 IAM을 사용하는 방법

IAM을 사용하여 X-Ray에 대한 액세스를 관리하려면 먼저 어떤 IAM 기능을 X-Ray와 함께 사용할 수 있는지를 이해해야 합니다. X-Ray 및 기타에서 IAM을 AWS 서비스 사용하는 방법을 개괄적으로 알아보려면 IAM 사용 설명서의 AWS 서비스 IAM으로 작업하기를 참조하세요.

AWS Identity and Access Management (IAM)를 사용하여 계정의 사용자 및 컴퓨팅 리소스에 X-Ray 권한을 부여할 수 있습니다. IAM은 API 수준에서 X-Ray 서비스에 대한 액세스를 제어하여 사용자가 사용하는 클라이언트(콘솔, AWS SDK AWS CLI)에 관계없이 권한을 균일하게 적용합니다.

X-Ray 콘솔을 사용하여 트레이스 맵과 세그먼트를 보려면 읽기 권한만 있으면 됩니다. 콘솔 액세스를 활성화하려면 `AWSXrayReadOnlyAccess` 관리형 정책을 IAM 사용자에 추가합니다.

로컬 개발 및 테스트를 위해서는 읽기 및 쓰기 권한을 갖는 IAM 역할을 생성합니다. 역할을 맡고 역할에 대한 임시 자격 증명을 저장합니다. 이러한 자격 증명을 X-Ray 데몬, AWS CLI및 AWS SDK와 함께 사용할 수 있습니다. 자세한 내용은 AWS CLI임시 보안 자격 증명 사용을 참조하세요.

계측된 앱을에 배포 AWS하려면 쓰기 권한이 있는 IAM 역할을 생성하고 애플리케이션을 실행하는 리소스에 할당합니다. `AWSXRayDaemonWriteAccess`에는 트레이스를 업로드할 수 있는 권한과 샘플링 규칙 사용을 지원하는 몇 가지 읽기 권한도 포함되어 있습니다.

읽기 및 쓰기 정책에는 암호화 키 설정 및 샘플링 규칙을 구성할 권한은 포함되지 않습니다. `AWSXrayFullAccess`을 사용하여 이러한 설정에 액세스하거나, 사용자 지정 정책에서 구성 API를 추가합니다. 생성하는 고객 관리형 키를 사용한 암호화 및 해독을 위해서는 해당 키를 사용할 권한도 필요합니다.

주제

- X-Ray identity-based 자격 증명 기반 정책
- X-Ray 리소스 기반 정책
- X-Ray 태그 기반 인증
- 로컬에서 애플리케이션 실행
- 에서 애플리케이션 실행 AWS
- 암호화에 대한 사용자 권한

X-Ray identity-based 자격 증명 기반 정책

IAM ID 기반 정책을 사용하면 허용되거나 거부되는 작업과 리소스뿐 아니라 작업이 허용되거나 거부되는 조건을 지정할 수 있습니다. X-Ray는 특정 작업, 리소스 및 조건 키를 지원합니다. JSON 정책에

서 사용하는 모든 요소에 대해 알고 싶다면 IAM 사용 설명서의 IAM JSON 정책 요소 참조를 참조하세요.

작업

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 위탁자가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

JSON 정책의 `Action` 요소는 정책에서 액세스를 허용하거나 거부하는 데 사용할 수 있는 작업을 설명합니다. 정책 작업은 일반적으로 연결된 AWS API 작업과 이름이 동일합니다. 일치하는 API 작업이 없는 권한 전용 작업 같은 몇 가지 예외도 있습니다. 정책에서 여러 작업이 필요한 몇 가지 작업도 있습니다. 이러한 추가 작업을 일컬어 종속 작업이라고 합니다.

연결된 작업을 수행할 수 있는 권한을 부여하기 위한 정책에 작업을 포함하세요.

X-Ray의 정책 작업은 작업 앞에 다음 접두사를 사용합니다. `xray:` 예를 들어 X-Ray `GetGroup` API 작업을 사용하여 그룹 리소스 세부 정보를 검색하는 권한을 부여하려면 해당 정책에 `xray:GetGroup` 작업을 포함합니다. 정책 문에는 `Action` 또는 `NotAction` 요소가 포함되어야 합니다. X-Ray는 이 서비스로 수행할 수 있는 작업을 설명하는 고유한 작업 세트를 정의합니다.

명령문 하나에 여러 태스크를 지정하려면 다음과 같이 쉼표로 구분합니다.

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

와일드카드(*)를 사용하여 여러 작업을 지정할 수 있습니다. 예를 들어, `Get`라는 단어로 시작하는 모든 태스크를 지정하려면 다음 태스크를 포함합니다.

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

X-Ray 작업 목록을 보려면 IAM 사용 설명서의 AWS X-Ray에서 정의한 작업을 참조하세요.

리소스

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

`Resource` JSON 정책 요소는 작업이 적용되는 하나 이상의 객체를 지정합니다. 문에는 `Resource`또는 `NotResource`요소가 반드시 추가되어야 합니다. 모범 사례에 따라 Amazon 리소스 이름(ARN)을

사용하여 리소스를 지정합니다. 리소스 수준 권한이라고 하는 특정 리소스 유형을 지원하는 작업에 대해 이를 수행할 수 있습니다.

작업 나열과 같이 리소스 수준 권한을 지원하지 않는 작업의 경우, 와일드카드(*)를 사용하여 해당 문이 모든 리소스에 적용됨을 나타냅니다.

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

A 정책을 사용하여 리소스에 대한 액세스를 제어할 수 있습니다. 리소스 수준의 권한을 지원하는 작업은 Amazon 리소스 이름(ARN)을 사용하여 정책이 적용되는 리소스를 확인할 수 있습니다.

IAM 정책에서 모든 X-Ray 작업을 사용하여 해당 작업을 사용하기 위한 사용자 권한을 부여하거나 거부할 수 있습니다. 하지만 모든 X-Ray 작업이 리소스 수준 권한을 지원하는 것은 아닙니다. 여기에서 리소스 수준 권한이란 작업이 가능한 리소스를 지정할 수 있는 권한을 말합니다.

리소스 수준 권한을 지원하지 않는 작업의 경우 "*"를 리소스로 사용해야 합니다.

다음 X-Ray 작업은 리소스 수준 권한을 지원합니다.

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

다음은 `CreateGroup` 작업에 대한 자격 증명 기반 권한 정책 예제입니다. 이 예제에서는 와일드카드로 고유 ID를 가지는 그룹 이름 `local-users`와 관련된 ARN의 사용을 보여줍니다. 고유 ID는 그룹을 만들 때 생성되기 때문에 정책에서 미리 예측할 수 없습니다. `GetGroup`, `UpdateGroup` 또는 `DeleteGroup`을 사용할 때, 이는 ID를 포함하여 와일드카드 또는 정확한 ARN으로 정의될 수 있습니다.

> **Note**
>
> 샘플링 규칙의 ARN은 규칙의 이름으로 정의됩니다. 그룹 ARN과 달리 샘플링 규칙은 따로 생성되는 고유 ID가 없습니다.

X-Ray 리소스 유형 및 해당 ARN의 목록을 보려면 IAM 사용 설명서의 AWS X-Ray에서 정의된 리소스를 참조하세요. 각 리소스의 ARN을 지정할 수 있는 작업을 알아보려면 AWS X-Ray가 정의한 작업을 참조하세요.

조건 키

X-Ray는 서비스별 조건 키를 제공하지 않지만, 일부 전역 조건 키 사용을 지원합니다. 모든 AWS 전역 조건 키를 보려면 IAM 사용 설명서의 AWS 전역 조건 컨텍스트 키를 참조하세요.

예시

X-Ray 자격 증명 기반 정책의 예를 보려면 AWS X-Ray 자격 증명 기반 정책 예제 단원을 참조하세요.

X-Ray 리소스 기반 정책

X-Ray는 Amazon SNS 활성 추적과 같은 현재 및 향후 AWS 서비스 통합을 위한 리소스 기반 정책을 지원합니다. X-Ray 리소스 기반 정책은 다른 AWS Management Console또는 AWS SDK 또는 CLI를 통해 업데이트할 수 있습니다. 예를 들어, Amazon SNS 콘솔은 X-Ray로 추적을 전송하기 위한 리소스 기반 정책을 자동으로 구성하려고 시도합니다. 다음 정책 문서에서는 X-Ray 리소스 기반 정책을 수동으로 구성하는 예제를 제공합니다.

Example Amazon SNS 활성 추적에 대한 X-Ray 리소스 기반 정책 예시

이 예시 정책 문서는 추적 데이터를 X-Ray로 전송하는 데 필요한 Amazon SNS의 권한을 지정합니다.

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

CLI를 사용하여 추적 데이터를 X-Ray로 전송할 수 있는 Amazon SNS 권한을 부여하는 리소스 기반 정책을 생성합니다.

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

이러한 예제를 사용하려면 *partition*, *account-id*, 및 *regiontopic-name*를 특정 AWS 파티션, 리전, 계정 ID 및 Amazon SNS 주제 이름으로 바꿉니다. 모든 Amazon SNS 주제가 추적 데이터를 X-Ray로 전송하도록 권한을 부여하려면 주제 이름을 *로 바꾸십시오.

X-Ray 태그 기반 인증

X-Ray 그룹이나 샘플링 규칙에 태그를 첨부하거나 요청에 포함된 태그를 X-Ray에 전달할 수 있습니다. 태그에 근거하여 액세스를 제어하려면 `xray:ResourceTag/`*key-name*, `aws:RequestTag/`*key-name* 또는 `aws:TagKeys` 조건 키를 사용하여 정책의 조건 요소에 태그 정보를 제공합니다. X-Ray 리소스 태깅에 대한 자세한 내용은 X-Ray 샘플링 규칙 및 그룹 태그 지정하기 단원을 참조하세요.

리소스의 태그를 기반으로 리소스에 대한 액세스를 제한하는 자격 증명 기반 정책의 예시는 태그에 기반한 X-Ray 그룹 및 샘플링 규칙에 대한 액세스 관리에서 확인할 수 있습니다.

로컬에서 애플리케이션 실행

계측된 애플리케이션은 추적 데이터를 X-Ray 대몬(daemon)으로 전송합니다. 대몬(daemon)은 세그먼트 문서를 버퍼링하여 X-Ray 서비스에 일괄적으로 업로드합니다. 대몬(daemon)이 추적 데이터와 원격 분석을 X-Ray 서비스에 업로드하려면 쓰기 권한이 필요합니다.

대몬(daemon)을 로컬에서 실행하는 경우 IAM 역할을 생성하고 역할을 수임한 다음 환경 변수나 사용자 폴더의 `.aws`라는 폴더 내 `credentials`라는 파일에 임시 자격 증명을 저장합니다. 자세한 내용은 AWS CLI임시 보안 자격 증명 사용을 참조하세요.

Example ~/.aws/credentials

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

AWS SDK 또는와 함께 사용할 자격 증명을 이미 구성한 경우 데몬 AWS CLI이 이를 사용할 수 있습니다. 여러 프로파일을 사용할 수 있을 경우 대몬(daemon)은 기본 프로파일을 사용합니다.

에서 애플리케이션 실행 AWS

에서 애플리케이션을 실행할 때 역할을 AWS사용하여 데몬을 실행하는 Amazon EC2 인스턴스 또는 Lambda 함수에 권한을 부여합니다.

- Amazon Elastic Compute Cloud(Amazon EC2) – IAM 역할을 생성하고 인스턴스 프로파일로 EC2 인스턴스에 연결합니다.
- Amazon Elastic Container Service(Amazon ECS) — IAM 역할을 생성하고 컨테이너 인스턴스에 컨테이너 인스턴스 IAM 역할로 연결합니다.
- AWS Elastic Beanstalk (Elastic Beanstalk) - Elastic Beanstalk는 기본 인스턴스 프로파일에 X-Ray 권한을 포함합니다. 기본 인스턴스 프로파일을 사용하거나 쓰기 권한을 사용자 지정 인스턴스 프로파일에 추가할 수 있습니다.
- AWS Lambda (Lambda) - 함수의 실행 역할에 쓰기 권한을 추가합니다.

X-Ray에 사용할 역할을 생성하려면

1. IAM 콘솔을 엽니다.
2. 역할을 선택합니다.
3. 새 역할 생성을 선택합니다.
4. Role Name(역할 이름)에 **`xray-application`**을 입력합니다. 다음 단계를 선택합니다.
5. [Role Type]에서 [Amazon EC2]를 선택합니다.
6. 관리형 정책을 연결하여 애플리케이션에 AWS 서비스서비스에 대한 액세스를 부여합니다.

- AWSXRayDaemonWriteAccess – X-Ray 대몬(daemon)에 추적 데이터를 업로드할 수 있는 권한을 부여합니다.

  애플리케이션이 AWS SDK를 사용하여 다른 서비스에 액세스하는 경우 해당 서비스에 대한 액세스 권한을 부여하는 정책을 추가합니다.

7. 다음 단계를 선택합니다.
8. 역할 생성을 선택합니다.

암호화에 대한 사용자 권한

X-Ray는 모든 트레이스 데이터를 기본적으로 암호화하며, 관리하는 키를 사용하도록 구성할 수 있습니다. AWS Key Management Service 고객 관리형 키를 선택하는 경우 키의 액세스 정책을 통해 X-Ray에 암호화에 사용할 수 있는 권한을 부여할 수 있는지 확인해야 합니다. 계정의 다른 사용자도 X-Ray 콘솔에서 암호화된 추적 데이터를 보려면 키에 대한 액세스 권한이 필요합니다.

고객 관리형 키의 경우 다음 작업을 허용하는 액세스 정책으로 키를 구성합니다.

- X-Ray에서 키를 구성하는 사용자는 `kms:CreateGrant` 및 `kms:DescribeKey`를 직접 호출할 수 있는 권한을 가집니다.
- 암호화된 추적 데이터에 액세스할 수 있는 사용자는 `kms:Decrypt`를 호출할 수 있는 권한을 가집니다.

IAM 콘솔의 키 구성 섹션에서 키 사용자 그룹에 사용자를 추가하면 해당 사용자는 이 두 가지 작업에 대한 권한을 갖게 됩니다. 권한은 키 정책에만 설정하면 되므로 사용자, 그룹 또는 역할에 대한 AWS KMS 권한이 필요하지 않습니다. 자세한 내용은 AWS KMS 개발자 안내서의 키 정책 사용을 참조하세요.

기본 암호화의 경우 또는 AWS 관리형 CMK(`aws/xray`)를 선택하는 경우 권한은 X-Ray APIs. `AWSXrayFullAccess`에 포함된 `PutEncryptionConfig`에 액세스할 수 있는 사람은 누구나 암호화 구성을 변경할 수 있습니다. 사용자가 암호화 키를 변경하지 못하게 하려면 `PutEncryptionConfig`를 사용할 권한을 부여하지 마십시오.

AWS X-Ray 자격 증명 기반 정책 예제

기본적으로 사용자 및 역할에는 X-Ray 리소스를 생성하거나 수정할 수 있는 권한이 없습니다. 또한 AWS Management Console AWS CLI또는 AWS API를 사용하여 작업을 수행할 수 없습니다. 관리자

는 지정된 리소스에서 특정 API 태스크를 수행할 수 있는 권한을 사용자와 역할에게 부여하는 IAM 정책을 생성해야 합니다. 그런 다음 관리자는 해당 권한이 필요한 사용자 또는 그룹에 이러한 정책을 연결해야 합니다.

이러한 예제 JSON 정책 문서를 사용하여 IAM 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서의 JSON 탭에서 정책 생성을 참조하세요.

주제

- 정책 모범 사례
- X-Ray 콘솔 사용하기
- 사용자가 자신이 권한을 볼 수 있도록 허용
- 태그에 기반한 X-Ray 그룹 및 샘플링 규칙에 대한 액세스 관리
- X-Ray에 대한 IAM 관리 정책
- AWS 관리형 정책에 대한 X-Ray 업데이트
- IAM 정책 내에서 리소스 지정

정책 모범 사례

ID 기반 정책에 따라 계정에서 사용자가 X-Ray 리소스를 생성, 액세스 또는 삭제할 수 있는지 여부가 결정됩니다. 이 작업으로 인해 AWS 계정에 비용이 발생할 수 있습니다. ID 기반 정책을 생성하거나 편집할 때는 다음 지침과 권장 사항을 따릅니다.

- AWS 관리형 정책을 시작하고 최소 권한으로 전환 - 사용자 및 워크로드에 권한 부여를 시작하려면 많은 일반적인 사용 사례에 대한 권한을 부여하는 AWS 관리형 정책을 사용합니다. 에서 사용할 수 있습니다 AWS 계정. 사용 사례에 맞는 AWS 고객 관리형 정책을 정의하여 권한을 추가로 줄이는 것이 좋습니다. 자세한 정보는 IAM 사용 설명서의 AWS 관리형 정책 또는 AWS 직무에 대한 관리형 정책을 참조하세요.
- 최소 권한 적용 – IAM 정책을 사용하여 권한을 설정하는 경우, 작업을 수행하는 데 필요한 권한만 부여합니다. 이렇게 하려면 최소 권한으로 알려진 특정 조건에서 특정 리소스에 대해 수행할 수 있는 작업을 정의합니다. IAM을 사용하여 권한을 적용하는 방법에 대한 자세한 정보는 IAM 사용 설명서에 있는 IAM의 정책 및 권한을 참조하세요.
- IAM 정책의 조건을 사용하여 액세스 추가 제한 – 정책에 조건을 추가하여 작업 및 리소스에 대한 액세스를 제한할 수 있습니다. 예를 들어, SSL을 사용하여 모든 요청을 전송해야 한다고 지정하는 정책 조건을 작성할 수 있습니다. AWS 서비스와 같은 특정를 통해 사용되는 경우 조건을 사용하여 서비스 작업에 대한 액세스 권한을 부여할 수도 있습니다 AWS CloudFormation. 자세한 정보는 IAM 사용 설명서의 IAM JSON 정책 요소: 조건을 참조하세요.

- IAM Access Analyzer를 통해 IAM 정책을 확인하여 안전하고 기능적인 권한 보장 - IAM Access Analyzer에서는 IAM 정책 언어(JSON)와 모범 사례가 정책에서 준수되도록 새로운 및 기존 정책을 확인합니다. IAM Access Analyzer는 100개 이상의 정책 확인 항목과 실행 가능한 추천을 제공하여 안전하고 기능적인 정책을 작성하도록 돕습니다. 자세한 내용은 IAM 사용 설명서의 IAM Access Analyzer에서 정책 검증을 참조하세요.
- 다중 인증(MFA) 필요 -에서 IAM 사용자 또는 루트 사용자가 필요한 시나리오가 있는 경우 추가 보안을 위해 MFA를 AWS 계정합니다. API 작업을 직접 호출할 때 MFA가 필요하면 정책에 MFA 조건을 추가합니다. 자세한 내용은 IAM 사용 설명서의 MFA를 통한 보안 API 액세스를 참조하세요.

IAM의 모범 사례에 대한 자세한 내용은 IAM 사용 설명서의 IAM의 보안 모범 사례를 참조하세요.

X-Ray 콘솔 사용하기

AWS X-Ray 콘솔에 액세스하려면 최소 권한 집합이 있어야 합니다. 이러한 권한은에서 X-Ray 리소스에 대한 세부 정보를 나열하고 볼 수 있도록 허용해야 합니다 AWS 계정. 최소 필수 권한보다 더 제한적인 ID 기반 정책을 생성하는 경우, 콘솔이 해당 정책에 연결된 엔티티(사용자 또는 역할)에 대해 의도대로 작동하지 않습니다.

이러한 엔터티가 여전히 X-Ray 콘솔을 사용할 수 있도록 하려면 `AWSXRayReadOnlyAccess` AWS 관리형 정책을 엔터티에 연결합니다. 이 정책은 X-Ray의 IAM 관리형 정책에 자세히 설명되어 있습니다. 자세한 내용은 IAM 사용 설명서의 사용자에게 권한 추가를 참조하십시오.

AWS CLI 또는 AWS API만 호출하는 사용자에게는 최소 콘솔 권한을 허용할 필요가 없습니다. 그 대신, 수행하려는 API 작업과 일치하는 작업에만 액세스할 수 있도록 합니다.

사용자가 자신이 권한을 볼 수 있도록 허용

이 예제는 IAM 사용자가 자신의 사용자 ID에 연결된 인라인 및 관리형 정책을 볼 수 있도록 허용하는 정책을 생성하는 방법을 보여줍니다. 이 정책에는 콘솔에서 또는 AWS CLI 또는 AWS API를 사용하여 프로그래밍 방식으로이 작업을 완료할 수 있는 권한이 포함됩니다.

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

태그에 기반한 X-Ray 그룹 및 샘플링 규칙에 대한 액세스 관리

ID 기반 정책의 조건을 사용하여 태그를 기반으로 X-Ray 그룹 및 샘플링 규칙에 대한 액세스를 제어할 수 있습니다. 다음 예제 정책은 사용자 역할에 대해 태그 `stage:prod` 또는 `stage:preprod`를 사용하여 그룹을 생성, 삭제 또는 업데이트할 수 있는 사용자 역할의 권한을 거부하는 데 사용할 수 있습니다. X-Ray 샘플링 규칙 및 그룹 태깅에 대한 자세한 내용은 X-Ray 샘플링 규칙 및 그룹 태그 지정하기을 참조하십시오.

샘플링 규칙 생성을 거부하려면 `aws:RequestTag`을 사용하여 생성 요청의 일부로 전달할 수 없는 태그를 표시하세요. 샘플링 규칙의 업데이트 또는 삭제를 거부하려면 `aws:ResourceTag`를 사용하여 해당 리소스의 태그에 기반한 작업을 거부하십시오.

계정 내 사용자에게 이러한 정책을 첨부(또는 단일 정책으로 결합한 다음 정책을 첨부)할 수 있습니다. 사용자가 그룹 또는 샘플링 규칙을 변경하려면 그룹 또는 샘플링 규칙에 `stage=prepod` 또는 `stage=prod` 태그를 지정해서는 안 됩니다. 조건 키 이름은 대소문자를 구분하지 않기 때문에 조건 태그 키 `Stage`는 `Stage` 및 `stage` 모두와 일치합니다. 조건 블록에 대한 자세한 내용은 IAM 사용 설명서의 IAM JSON 정책 요소: 조건을 참조하세요.

다음 정책이 연결된 역할을 가진 사용자는 리소스에 `role:admin` 태그를 추가할 수 없으며 `role:admin`가 연결된 리소스에서 태그를 제거할 수 없습니다.

JSON

```
{
    "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-Ray에 대한 IAM 관리 정책

권한 부여를 간소화하기 위해 IAM은 각 서비스에 대해 관리형 정책을 지원합니다. 서비스는 새 APIs. 는 읽기 전용, 쓰기 전용 및 관리자 사용 사례에 대한 관리형 정책을 AWS X-Ray 제공합니다.

- `AWSXrayReadOnlyAccess` - X-Ray 콘솔 AWS CLI또는 AWS SDK를 사용하여 X-Ray API에서 트레이스 데이터, 트레이스 맵, 인사이트 및 X-Ray 구성을 가져올 수 있는 읽기 권한입니다. 콘솔에서 CloudWatch 교차 계정 통합 관찰성의 일부로 소스 계정에서 공유된 추적을 볼 수 있도록 하는 통합 관찰성 액세스 관리자(OAM) `oam:ListSinks` 및 `oam:ListAttachedSinks` 권한이 포함되어 있습니다. `BatchGetTraceSummaryById` 및 `GetDistinctTraceGraphs` API 작업은 코드에서 호출하기 위한 것이 아니며 AWS CLI 및 AWS SDKs에 포함되지 않습니다.

```
{
    "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` - X-Ray 데몬 AWS CLI또는 AWS SDK를 사용하여 세그먼트 문서 및 원격 측정을 X-Ray API에 업로드할 수 있는 쓰기 권한입니다. 샘플링 규칙을 가져오고 샘플링 결과를 보고할 읽기 권한이 포함됩니다.

  JSON

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

- `AWSXrayCrossAccountSharingConfiguration` – 계정 간에 X-Ray 리소스를 공유하기 위한 Observability Access Manager 링크를 만들고, 관리하고, 볼 수 있는 권한을 부여합니다. 소스 계정과 모니터링 계정 간의 CloudWatch 계정 간 관찰성을 활성화하는 데 사용됩니다.

  JSON

```
{
    "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` – 읽기 권한, 쓰기 권한 및 암호화 키 설정과 샘플링 규칙을 구성할 권한을 포함하여 모든 X-Ray API를 사용할 권한입니다. 콘솔에서 CloudWatch 교차 계정 통합 관찰성의 일부로 소스 계정에서 공유된 추적을 볼 수 있도록 하는 통합 관찰성 액세스 관리자(OAM) `oam:ListSinks` 및 `oam:ListAttachedSinks` 권한이 포함되어 있습니다.

  JSON

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

```
        {
            "Effect": "Allow",
            "Action": [
                "xray:*",
                "oam:ListSinks"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "oam:ListAttachedLinks"
            ],
            "Resource": "arn:aws:oam:*:*:sink/*"
        }
    ]
}
```

관리형 정책을 IAM 사용자 그룹 또는 역할에 추가하는 방법

1. IAM 콘솔을 엽니다.
2. 인스턴스 프로파일, IAM 사용자 또는 IAM 사용자 그룹과 연결된 역할을 엽니다.
3. 권한에서 관리형 정책을 연결합니다.

AWS 관리형 정책에 대한 X-Ray 업데이트

이 서비스가 이러한 변경 사항을 추적하기 시작한 이후부터 X-Ray의 AWS 관리형 정책 업데이트에 대한 세부 정보를 봅니다. 이 페이지의 변경 사항에 대한 자동 알림을 받으려면 X-Ray 문서 기록 페이지에서 RSS 피드를 구독하세요.

변경 사항	설명	날짜
X-Ray용 IAM 관리형 정책 — 새로운 `AWSXrayCrossAccountSharingConfiguration` 및 업	X-Ray는 콘솔에서 CloudWatch 교차 계정 통합 관찰성의 일부로 소스 계정에서 공유된 추적을 볼 수 있도록 이러한 정책에 통합 관찰	2022년 11월 27일

변경 사항	설명	날짜
데이트된 AWSXrayReadOnlyAccess 및 AWSXrayFullAccess 정책이 추가되었습니다.	성 액세스 관리자(OAM) 권한 oam:ListSinks 및 oam:ListAttachedSinks 가 추가되었습니다.	
X-Ray용 IAM 관리형 정책 — AWSXrayReadOnlyAccess 정책 업데이트	X-Ray가 ListResourcePolicies API 작업을 추가했습니다.	2022년 11월 15일
X-Ray 콘솔 사용 — AWSXrayReadOnlyAccess 정책 업데이트	X-Ray는 두 개의 새로운 API 작업 BatchGetTraceSummaryById 과 GetDistinctTraceGraphs 를 추가했으며, 이 작업들은 코드로 호출되는 것이 아닙니다. 따라서 이러한 API 작업은 AWS CLI 및 AWS SDKs에 포함되지 않습니다.	2022년 11월 11일

IAM 정책 내에서 리소스 지정

IAM 정책을 사용하여 리소스에 대한 액세스를 제어할 수 있습니다. 리소스 수준의 권한을 지원하는 작업은 Amazon 리소스 이름(ARN)을 사용하여 정책이 적용되는 리소스를 확인할 수 있습니다.

IAM 정책에서 모든 X-Ray 작업을 사용하여 해당 작업을 사용하기 위한 사용자 권한을 부여하거나 거부할 수 있습니다. 하지만 모든 X-Ray 작업이 리소스 수준 권한을 지원하는 것은 아닙니다. 여기에서 리소스 수준 권한이란 작업이 가능한 리소스를 지정할 수 있는 권한을 말합니다.

리소스 수준 권한을 지원하지 않는 작업의 경우 "*"를 리소스로 사용해야 합니다.

다음 X-Ray 작업은 리소스 수준 권한을 지원합니다.

- CreateGroup
- GetGroup
- UpdateGroup

- DeleteGroup
- CreateSamplingRule
- UpdateSamplingRule
- DeleteSamplingRule

다음은 CreateGroup 작업에 대한 자격 증명 기반 권한 정책 예제입니다. 이 예제에서는 와일드카드로 고유 ID를 가지는 그룹 이름 local-users와 관련된 ARN의 사용을 보여줍니다. 고유 ID는 그룹을 만들 때 생성되기 때문에 정책에서 미리 예측할 수 없습니다. GetGroup, UpdateGroup 또는 DeleteGroup을 사용할 때, 이는 ID를 포함하여 와일드카드 또는 정확한 ARN으로 정의될 수 있습니다.

JSON

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

다음은 CreateSamplingRule 작업에 대한 자격 증명 기반 권한 정책 예제입니다.

JSON

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

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

> **Note**
>
> 샘플링 규칙의 ARN은 규칙의 이름으로 정의됩니다. 그룹 ARN과 달리 샘플링 규칙은 따로 생성되는 고유 ID가 없습니다.

자격 AWS X-Ray 증명 및 액세스 문제 해결

다음 정보를 사용하여 X-Ray 및 IAM에서 발생할 수 있는 공통적인 문제를 진단하고 수정할 수 있습니다.

주제

- X-Ray에서 작업을 수행할 권한이 없음
- iam:PassRole을 수행할 권한이 없음
- 관리자인데, 다른 사용자가 X-Ray에 액세스할 수 있게 허용하려고 합니다
- 내 외부의 사람이 내 X-Ray 리소스에 액세스 AWS 계정 하도록 허용하고 싶습니다.

X-Ray에서 작업을 수행할 권한이 없음

에서 작업을 수행할 수 있는 권한이 없다는 AWS Management Console 메시지가 표시되면 관리자에게 문의하여 도움을 받아야 합니다. 관리자는 로그인 보안 인증 정보를 제공한 사람입니다.

다음 예의 오류는 `mateojackson` 사용자가 콘솔을 사용하여 샘플링 규칙에 대한 세부 정보를 보려고 하지만 `xray:GetSamplingRules` 권한이 없는 경우에 발생합니다.

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

이 경우 Mateo는 `xray:GetSamplingRules` 작업을 사용하여 샘플링 규칙 리소스에 대한 액세스를 허용하도록 정책을 업데이트할 것을 관리자에게 요청합니다.

iam:PassRole을 수행할 권한이 없음

`iam:PassRole` 작업을 수행할 수 있는 권한이 없다는 오류가 수신되면 X-Ray에 역할을 전달할 수 있도록 정책을 업데이트해야 합니다.

일부 AWS 서비스 에서는 새 서비스 역할 또는 서비스 연결 역할을 생성하는 대신 기존 역할을 해당 서비스에 전달할 수 있습니다. 이렇게 하려면 사용자가 서비스에 역할을 전달할 수 있는 권한을 가지고 있어야 합니다.

다음 예제 오류는 `marymajor`(이)라는 IAM 사용자가 콘솔을 사용하여 X-Ray에서 작업을 수행하려고 하는 경우에 발생합니다. 하지만 작업을 수행하려면 서비스 역할이 부여한 권한이 서비스에 있어야 합니다. Mary는 서비스에 역할을 전달할 수 있는 권한을 가지고 있지 않습니다.

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

이 경우, Mary가 `iam:PassRole` 작업을 수행할 수 있도록 Mary의 정책을 업데이트해야 합니다.

도움이 필요한 경우 AWS 관리자에게 문의하세요. 관리자는 로그인 자격 증명을 제공한 사람입니다.

관리자인데, 다른 사용자가 X-Ray에 액세스할 수 있게 허용하려고 합니다

다른 사용자가 X-Ray에 액세스할 수 있도록 허용하려면 액세스가 필요한 사용자 또는 애플리케이션에 권한을 부여해야 합니다. AWS IAM Identity Center 를 사용하여 사용자 및 애플리케이션을 관리하는 경우 사용자 또는 그룹에 권한 세트를 할당하여 액세스 수준을 정의합니다. 권한 세트는 IAM 정책을 자동으로 생성하고 사용자 또는 애플리케이션과 연결된 IAM 역할에 할당합니다. 자세한 내용은 AWS IAM Identity Center 사용 설명서에서 권한 세트를 참조하세요.

IAM Identity Center를 사용하지 않는 경우 액세스가 필요한 사용자 또는 애플리케이션에 대한 IAM 엔터티(사용자 또는 역할)를 생성해야 합니다. 그런 다음 X-ray에 대한 올바른 권한을 부여하는 정책을 엔터티에 연결해야 합니다. 권한이 부여되면 사용자 또는 애플리케이션 개발자에게 자격 증명을 제공합니다. 이들은 이 자격 증명을 사용하여 AWS에 액세스합니다. IAM 사용자, 그룹, 정책, 권한 생성에 대한 자세한 내용은 IAM 사용 설명서의 IAM ID 및 IAM 정책과 권한을 참조하세요.

내 외부의 사람이 내 X-Ray 리소스에 액세스 AWS 계정 하도록 허용하고 싶습니다.

다른 계정의 사용자 또는 조직 외부의 사람이 리소스에 액세스할 때 사용할 수 있는 역할을 생성할 수 있습니다. 역할을 수임할 신뢰할 수 있는 사람을 지정할 수 있습니다. 리소스 기반 정책 또는 액세스 제

어 목록(ACL)을 지원하는 서비스의 경우, 이러한 정책을 사용하여 다른 사람에게 리소스에 대한 액세스 권한을 부여할 수 있습니다.

자세히 알아보려면 다음을 참조하세요.

- X-Ray에서 이러한 기능을 지원하는지 여부를 알아보려면 AWS X-Ray 에서 IAM을 사용하는 방법 단원을 참조하세요.
- 소유 AWS 계정 한의 리소스에 대한 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 AWS 계정 소유한 다른의 IAM 사용자에게 액세스 권한 제공을 참조하세요.
- 리소스에 대한 액세스 권한을 타사에 제공하는 방법을 알아보려면 IAM 사용 설명서의 타사 AWS 계정 소유의에 액세스 권한 제공을 AWS 계정참조하세요.
- ID 페더레이션을 통해 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 외부에서 인증된 사용자에게 액세스 권한 제공(ID 페더레이션)을 참조하세요.
- 크로스 계정 액세스에 대한 역할과 리소스 기반 정책 사용의 차이점을 알아보려면 IAM 사용 설명서의 IAM의 크로스 계정 리소스 액세스를 참조하세요.

에서 로깅 및 모니터링 AWS X-Ray

모니터링은 AWS 솔루션의 안정성, 가용성 및 성능을 유지하는 중요한 역할을 합니다. 다중 지점 장애가 발생할 경우 보다 쉽게 디버깅할 수 있도록 AWS 솔루션의 모든 부분에서 모니터링 데이터를 수집해야 합니다.는 X-Ray 리소스를 모니터링하고 잠재적 인시던트에 대응하기 위한 여러 도구를 AWS 제공합니다.

AWS CloudTrail 로그

AWS X-Ray 는와 통합되어 X-Ray에서 사용자, 역할 또는 AWS 서비스가 수행한 API 작업을 AWS CloudTrail 기록합니다. CloudTrail을 사용하여 X-Ray API 요청을 실시간으로 모니터링하고 Amazon S3, Amazon CloudWatch Logs 및 Amazon CloudWatch Events에 로그를 저장할 수 있습니다. 자세한 내용은 를 사용하여 X-Ray API 호출 로깅 AWS CloudTrail 단원을 참조하십시오.

AWS Config 추적

AWS X-Ray 는와 통합되어 X-Ray 암호화 리소스에 대한 구성 변경을 AWS Config 기록합니다. AWS Config 를 사용하여 X-Ray 암호화 리소스를 인벤토리화하고, X-Ray 구성 기록을 감사하고, 리소스 변경 사항에 따라 알림을 보낼 수 있습니다. 자세한 내용은 를 사용하여 X-Ray 암호화 구성 변경 사항 추적 AWS Config 단원을 참조하십시오.

Amazon CloudWatch 모니터링

Java용 X-Ray SDK를 사용하여 수집된 X-Ray 세그먼트에서 샘플링되지 않은 Amazon CloudWatch 지표를 게시할 수 있습니다. 이러한 지표는 세그먼트의 시작 및 종료 시간과 오류, 장애 및 스로틀된 상태 플래그에서 파생됩니다. 이러한 트레이스 지표를 사용하여 하위 세그먼트 내에서 재시도 및 종속성 문제를 표시할 수 있습니다. 자세한 내용은 AWS X-Ray Java용 X-Ray SDK에 대한 지표 단원을 참조하십시오.

에 대한 규정 준수 검증 AWS X-Ray

AWS 서비스 가 특정 규정 준수 프로그램의 범위 내에 있는지 알아보려면 AWS 서비스 규정 준수 프로그램 제공 범위 섹션을 참조하고 관심 있는 규정 준수 프로그램을 선택합니다. 일반 정보는 AWS 규정 준수 프로그램.

를 사용하여 타사 감사 보고서를 다운로드할 수 있습니다 AWS Artifact. 자세한 내용은 에서 보고서 다운로드 AWS Artifact에서 .

사용 시 규정 준수 책임은 데이터의 민감도, 회사의 규정 준수 목표 및 관련 법률과 규정에 따라 AWS 서비스 결정됩니다.는 규정 준수를 지원하기 위해 다음 리소스를 AWS 제공합니다.

- 보안 규정 준수 및 거버넌스 - 이러한 솔루션 구현 가이드에서는 아키텍처 고려 사항을 설명하고 보안 및 규정 준수 기능을 배포하는 단계를 제공합니다.
- HIPAA 적격 서비스 참조 - HIPAA 적격 서비스가 나열되어 있습니다. 모두 HIPAA 자격이 AWS 서비스 있는 것은 아닙니다.
- AWS 규정 준수 리소스 -이 워크북 및 가이드 모음은 업계 및 위치에 적용될 수 있습니다.
- AWS 고객 규정 준수 가이드 - 규정 준수의 관점에서 공동 책임 모델을 이해합니다. 이 가이드에는 여러 프레임워크(미국 국립표준기술연구소(NIST), 결제 카드 산업 보안 표준 위원회(PCI), 국제표준화기구(ISO) 포함)의 보안 제어에 대한 지침을 보호하고 AWS 서비스 매핑하는 모범 사례가 요약되어 있습니다.
- AWS Config 개발자 안내서의 규칙을 사용하여 리소스 평가 -이 AWS Config 서비스는 리소스 구성이 내부 관행, 업계 지침 및 규정을 얼마나 잘 준수하는지 평가합니다.
- AWS Security Hub - 이를 AWS 서비스 통해 내 보안 상태를 포괄적으로 볼 수 있습니다 AWS. Security Hub는 보안 컨트롤을 사용하여 AWS 리소스를 평가하고 보안 업계 표준 및 모범 사례에 대한 규정 준수를 확인합니다. 지원되는 서비스 및 제어 목록은 Security Hub 제어 참조를 참조하세요.
- Amazon GuardDuty - 의심스러운 악의적인 활동이 있는지 환경을 모니터링하여 사용자, AWS 계정 워크로드, 컨테이너 및 데이터에 대한 잠재적 위협을 AWS 서비스 탐지합니다. GuardDuty는 특정 규

정 준수 프레임워크에서 요구하는 침입 탐지 요구 사항을 충족하여 PCI DSS와 같은 다양한 규정 준수 요구 사항을 따르는 데 도움을 줄 수 있습니다.

- AWS Audit Manager - 이를 AWS 서비스 통해 AWS 사용량을 지속적으로 감사하여 위험과 규정 및 업계 표준 준수를 관리하는 방법을 간소화할 수 있습니다.

의 복원성 AWS X-Ray

AWS 글로벌 인프라는 AWS 리전 및 가용 영역을 중심으로 구축됩니다.는 지연 시간이 짧고 처리량이 높으며 중복성이 높은 네트워킹과 연결된 물리적으로 분리되고 격리된 여러 가용 영역을 AWS 리전 제공합니다. 가용 영역을 사용하면 중단 없이 가용 영역 간에 자동으로 장애 조치가 이루어지는 애플리케이션 및 데이터베이스를 설계하고 운영할 수 있습니다. 가용 영역은 기존의 단일 또는 복수 데이터 센터 인프라보다 가용성, 내결함성, 확장성이 뛰어납니다.

AWS 리전 및 가용 영역에 대한 자세한 내용은 AWS 글로벌 인프라를 참조하세요.

의 인프라 보안 AWS X-Ray

관리형 서비스인는 AWS 글로벌 네트워크 보안으로 보호 AWS X-Ray 됩니다. AWS 보안 서비스 및가 인프라를 AWS 보호하는 방법에 대한 자세한 내용은 AWS 클라우드 보안을 참조하세요. 인프라 보안 모범 사례를 사용하여 AWS 환경을 설계하려면 보안 원칙 AWS Well-Architected Framework의 인프라 보호를 참조하세요.

AWS 에서 게시한 API 호출을 사용하여 네트워크를 통해 X-Ray에 액세스합니다. 고객은 다음을 지원해야 합니다.

- Transport Layer Security(TLS) TLS 1.2는 필수이며 TLS 1.3을 권장합니다.
- DHE(Ephemeral Diffie-Hellman) 또는 ECDHE(Elliptic Curve Ephemeral Diffie-Hellman)와 같은 완전 전송 보안(PFS)이 포함된 암호 제품군 Java 7 이상의 최신 시스템은 대부분 이러한 모드를 지원합니다.

또한 요청은 액세스 키 ID 및 IAM 위탁자와 관련된 보안 암호 액세스 키를 사용하여 서명해야 합니다. 또는 AWS Security Token Service(AWS STS)를 사용하여 임시 자격 증명을 생성하여 요청에 서명할 수 있습니다.

VPC 엔드포인트 AWS X-Ray 와 함께 사용

Amazon Virtual Private Cloud(VPC)를 사용하여 AWS 리소스를 호스팅하는 경우 VPC와 X-Ray 간에 프라이빗 연결을 설정할 수 있습니다. 이 연결을 사용하면 X-Ray 서비스가 퍼블릭 인터넷을 통하지 않고 Amazon VPC의 리소스와 통신할 수 있게 해줍니다.

Amazon VPC는 정의 AWS 서비스 한 가상 네트워크에서 AWS 리소스를 시작하는 데 사용할 수 있는 입니다. VPC를 사용하여 IP 주소 범위, 서브넷, 라우팅 테이블, 네트워크 게이트웨이 등의 네트워크 설정을 제어할 수 있습니다. VPC를 X-Ray에 연결하려면 인터페이스 VPC 엔드포인트를 정의하십시오. 이 엔드포인트를 이용하면 인터넷 게이트웨이나 Network Address Translation(NAT) 인스턴스, 또는 VPN 연결 없이도 X-Ray에 안정적이고 확장 가능하게 연결됩니다. 자세한 내용은 Amazon VPC 사용 설명서의 Amazon VPC란 무엇입니까를 참조하세요.

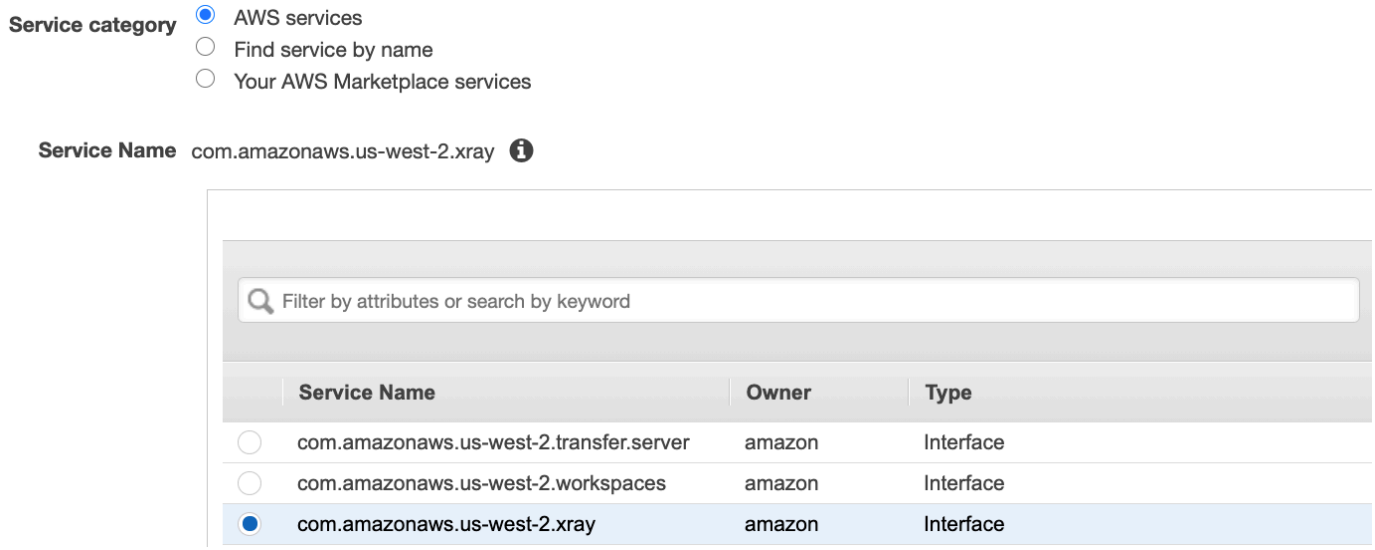
인터페이스 VPC 엔드포인트는 프라이빗 IP 주소와 함께 탄력적 네트워크 인터페이스를 사용하여 간의 프라이빗 통신을 지원하는 AWS 기술인 AWS PrivateLink AWS 서비스 로 구동됩니다. 자세한 내용은 New – AWS PrivateLink for AWS 서비스 블로그 게시물과 Amazon VPC 사용 설명서의 시작하기를 참조하세요.

선택한에서 X-Ray용 VPC 엔드포인트를 생성할 수 있도록 하려면 섹션을 AWS 리전참조하세요지원되는 리전.

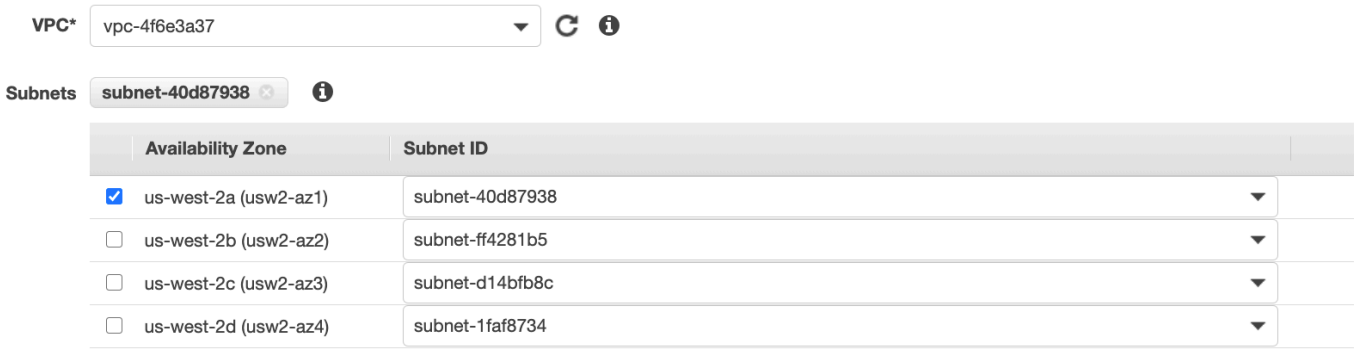
X-Ray용 VPC 엔드포인트 생성

VPC에서 X-Ray를 사용하려면 X-Ray용 인터페이스 VPC 엔드포인트를 생성하세요.

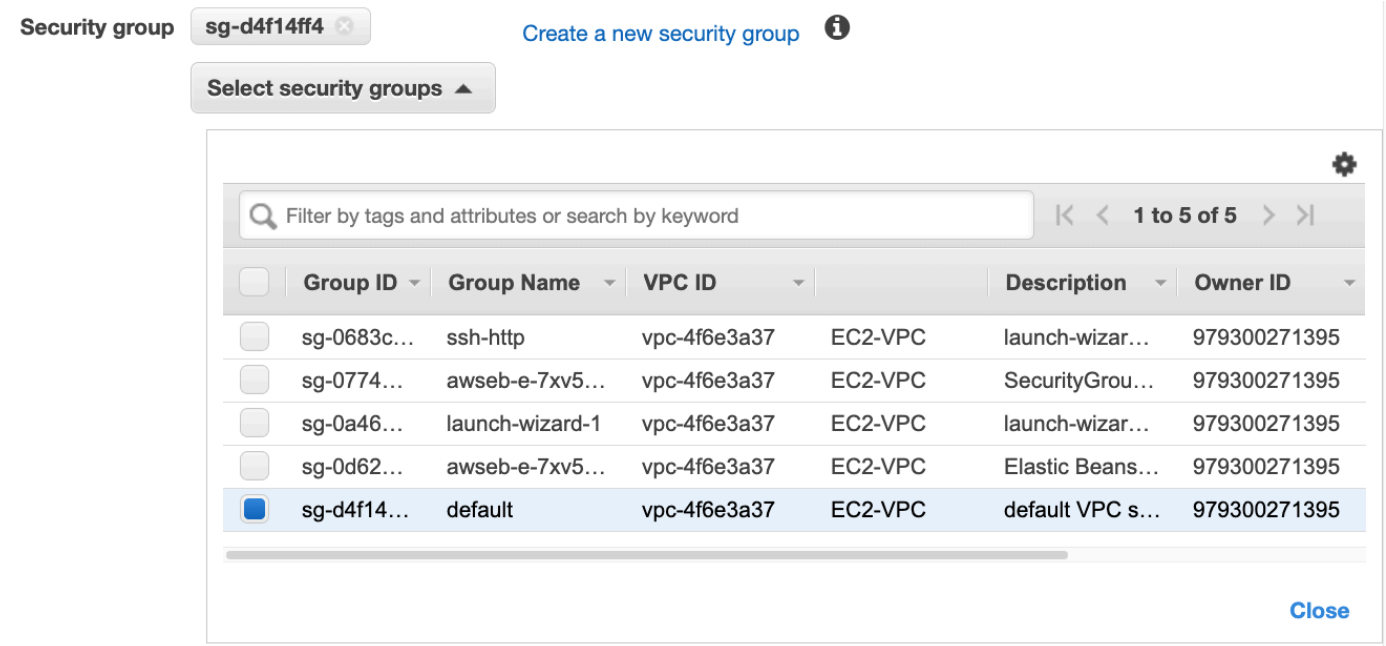
1. https://console.aws.amazon.com/vpc/에서 Amazon VPC 콘솔을 엽니다.
2. 탐색 창에서 엔드포인트로 이동한 다음 엔드포인트 생성을 선택합니다.
3. 를 검색하고 AWS X-Ray 서비스 이름을 선택합니다`com.amazonaws.`*`region`*`.xray`.



4. 원하는 VPC를 선택한 다음 인터페이스 엔드포인트를 사용할 VPC의 서브넷을 선택합니다. 선택한 서브넷에서 엔드포인트 네트워크 인터페이스가 생성됩니다. 각기 다른 가용 영역(서비스에 의해 지원)에서 하나 이상의 서브넷을 지정하여 인터페이스 엔드포인트가 가용 영역 장애 발생 시 복원을 지원합니다. 이렇게 하면 지정한 각 서브넷에 인터페이스 네트워크 인터페이스가 생성됩니다.



5. (선택 사항) 개인 DNS는 기본적으로 엔드포인트에 대해 활성화되어 있으므로 기본 DNS 호스트 이름을 사용하여 X-Ray에 요청할 수 있습니다. 필요에 따라 비활성화하도록 선택할 수 있습니다.
6. 보안 그룹을 지정하여 엔드포인트 네트워크 인터페이스와 연결합니다.



7. (선택 사항) X-Ray 서비스에 액세스할 수 있는 권한을 제어하는 사용자 지정 정책을 지정합니다. 기본적으로 전체 액세스 권한이 허용됩니다.

X-Ray VPC 엔드포인트에 대한 액세스 제어

VPC 엔드포인트 정책은 엔드포인트를 만들거나 수정 시 엔드포인트에 연결하는 IAM 리소스 정책입니다. 엔드포인트를 생성할 때 정책을 연결하지 않으면 Amazon VPC는 서비스에 대한 전체 액세스를 허용하는 기본 정책을 자동으로 연결합니다. 엔드포인트 정책은 IAM 사용자 정책 또는 서비스별 정책을 재정의하거나 대체하지 않습니다. 이는 엔드포인트에서 지정된 서비스로의 액세스를 제어하기 위한 별도의 정책입니다. 엔드포인트 정책은 JSON 형식으로 작성해야 합니다. 자세한 내용은 Amazon VPC 사용 설명서의 VPC 엔드포인트를 통해 서비스에 대한 액세스 제어를 참조하세요.

VPC 엔드포인트 정책을 통해 다양한 X-Ray 작업에 대한 권한을 제어할 수 있습니다. 예를 들어 PutTraceSegment만 허용하고 다른 모든 작업은 거부하도록 정책을 생성할 수 있습니다. 이렇게 하면 VPC의 워크로드 및 서비스가 추적 데이터만 X-Ray로 전송하고 데이터 검색, 암호화 구성 변경, 그룹 생성/업데이트와 같은 다른 작업은 거부하도록 제한됩니다.

다음은 X-Ray에 대한 엔드포인트 정책의 예입니다. 이 정책은 사용자가 VPC를 통해 X-Ray에 연결하여 X-Ray로 지표 데이터를 전송할 수 있도록 허용하고, 다른 X-Ray 작업을 수행하지 못하게 금지합니다.

```
{"Statement": [
    {"Sid": "Allow PutTraceSegments",
```

```
      "Principal": "*",
      "Action": [
        "xray:PutTraceSegments"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

X-Ray에 대한 VPC 엔드포인트 정책을 편집하려면

1. https://console.aws.amazon.com/vpc/에서 Amazon VPC 콘솔을 엽니다.
2. 탐색 창에서 엔드포인트를 선택합니다.
3. X-Ray용 엔드포인트를 아직 생성하지 않은 경우 X-Ray용 VPC 엔드포인트 생성의 과정을 따르십시오.
4. com.amazonaws.***region***.xray 엔드포인트를 선택한 다음 정책 탭을 선택합니다.
5. 정책 편집을 선택한 다음 변경합니다.

지원되는 리전

X-Ray는 현재 AWS 리전다음과 같은 VPC 엔드포인트를 지원합니다.

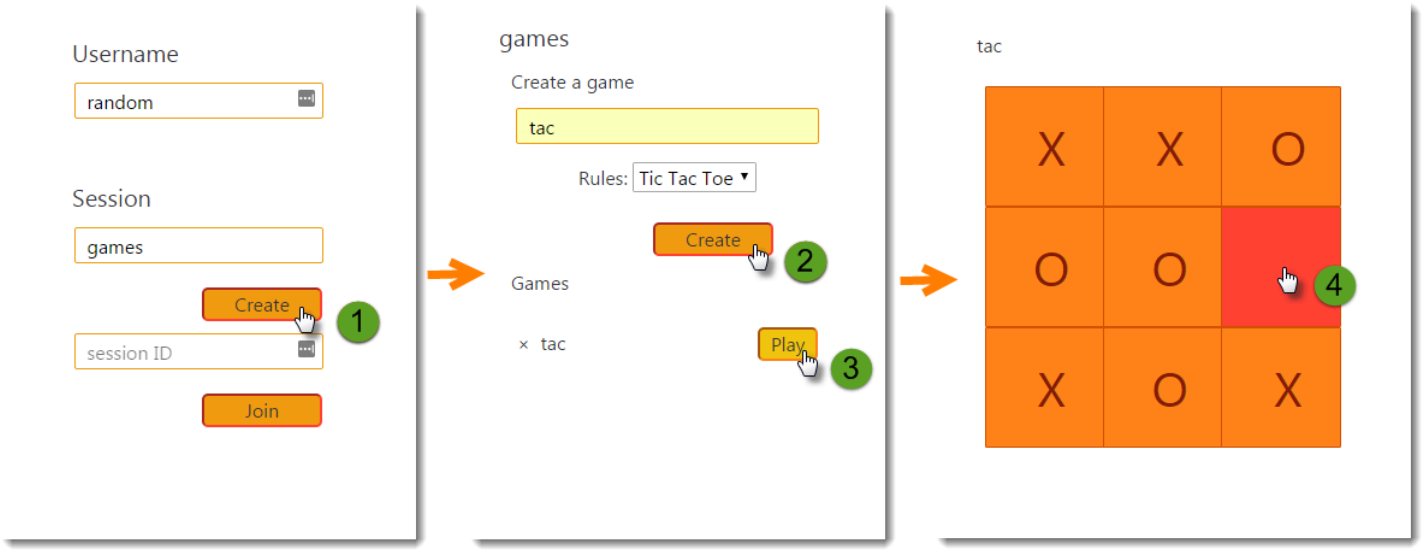
- 미국 동부(오하이오)
- 미국 동부(버지니아 북부)
- 미국 서부(캘리포니아 북부)
- 미국 서부(오레곤)
- 아프리카(케이프타운)
- 아시아 태평양(홍콩)
- 아시아 태평양(뭄바이)
- 아시아 태평양(오사카)
- 아시아 태평양(서울)
- 아시아 태평양(싱가포르)
- 아시아 태평양(시드니)
- 아시아 태평양(도쿄)

- 캐나다(중부)
- 유럽(프랑크푸르트)
- 유럽(아일랜드)
- 유럽(런던)
- 유럽(밀라노)
- 유럽(파리)
- 유럽(스톡홀름)
- 중동(바레인)
- 남아메리카(상파울루)
- AWS GovCloud(미국 동부)
- AWS GovCloud(미국 서부)

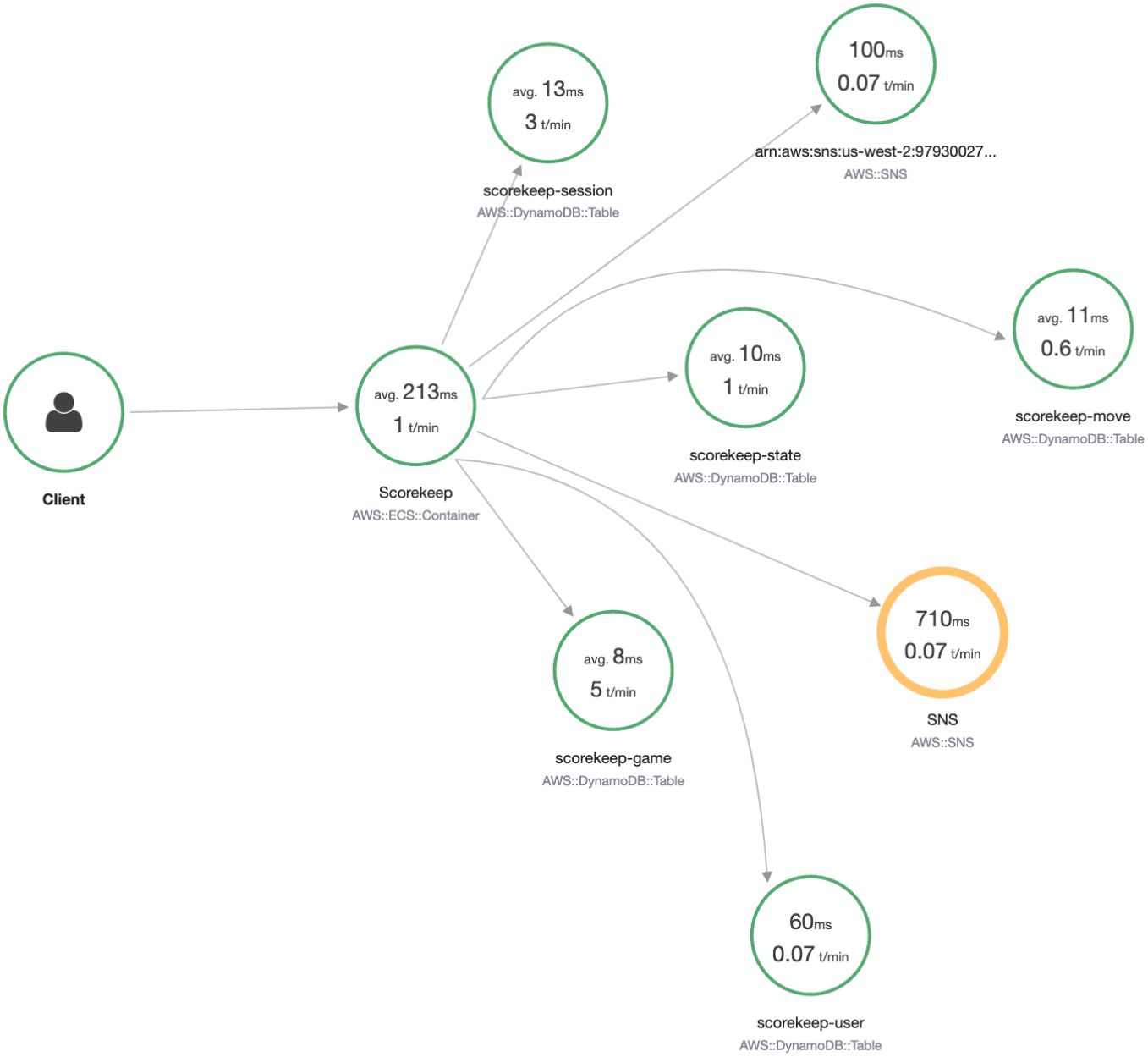
AWS X-Ray 샘플 애플리케이션

GitHub에서 사용할 수 있는 AWS X-Ray eb-java-scorekeep 샘플 앱은 AWS X-Ray SDK를 사용하여 수신 HTTP 호출, DynamoDB SDK 클라이언트 및 HTTP 클라이언트를 계측하는 방법을 보여줍니다. 샘플 앱은 AWS CloudFormation 를 사용하여 DynamoDB 테이블을 생성하고, 인스턴스에서 Java 코드를 컴파일하고, 추가 구성 없이 X-Ray 데몬을 실행합니다.

AWS Management Console 또는 를 사용하여 계측된 샘플 애플리케이션의 설치 및 사용을 시작하려면 Scorekeep 자습서를 참조하세요 AWS CLI.



여기에는 프런트엔드 웹 앱, 이 앱이 직접 호출하는 API, 이 앱이 데이터를 저장하는 데 사용하는 DynamoDB 테이블이 포함되어 있습니다. 필터, 플러그인 및 계측된 AWS SDK 클라이언트를 사용한 기본 계측은 프로젝트의 `xray-gettingstarted`브랜치에 표시됩니다. 이는 시작하기 자습서에서 배포하는 분기입니다. 이 분기에는 기본 사항만 포함되어 있으므로 `master` 분기와의 차이점을 확인하여 기본 사항을 신속하게 이해할 수 있습니다.



샘플 애플리케이션은 다음 파일에서 기본 구성을 보여 줍니다.

- HTTP 요청 필터 – `WebConfig.java`
- AWS SDK 클라이언트 계측 - `build.gradle`

애플리케이션의 `xray` 분기는 HTTPClient, 주석, SQL 쿼리, 사용자 지정 하위 세그먼트, 구성된 AWS Lambda 함수 및 구성된 초기화 코드 및 스크립트 사용을 포함합니다.

브라우저에서 사용자 로그인 및 AWS SDK for JavaScript 사용을 지원하기 위해 `xray-cognito`브랜치는 사용자 인증 및 권한 부여를 지원하기 위해 Amazon Cognito를 추가합니다. Amazon Cognito에서 가져온 보안 인증 정보를 사용하여 웹 앱은 트레이스 데이터를 X-Ray에 보내 클라이언트 관점에서의 요청 정보를 레코딩합니다. 브라우저 클라이언트는 트레이스 맵에 자체 노드로 표시되며, 사용자가 보고 있는 페이지의 URL 및 사용자의 ID를 포함한 추가 정보를 기록합니다.

마지막으로 `xray-worker`분기는 Amazon SQS 대기열에서 처리 중인 항목을 독립적으로 실행하는 계기화된 Python Lambda 함수를 추가합니다. Scorekeep는 게임이 끝날 때마다 대기열에 항목을 추가합니다. Cloudwatch 이벤트에 의해 트리거되는 Lambda 작업자는 몇 분 간격으로 대기열에서 항목을 가져온 후 처리하여 분석을 위해 게임 레코드를 Amazon S3에 저장합니다.

주제

- Scorekeep 샘플 애플리케이션 시작하기
- AWS SDK 클라이언트 수동 계측
- 추가 하위 세그먼트 생성
- 주석, 메타데이터 및 사용자 ID 기록
- 발신 HTTP 호출 구성
- PostgreSQL 데이터베이스에 대한 호출 구성
- AWS Lambda 함수 계측
- 시작 코드 구성
- 스크립트 구성
- 웹 앱 클라이언트 구성
- 작업자 스레드에서 구성된 클라이언트 사용

Scorekeep 샘플 애플리케이션 시작하기

이 자습서에서는를 사용하여 Amazon ECS에서 샘플 애플리케이션 및 X-Ray 데몬을 실행하는 리소스를 생성하고 구성하는 Scorekeep 샘플 애플리케이션의 `xray-gettingstarted`브랜치를 사용합니다. AWS CloudFormation 애플리케이션은 Spring 프레임워크를 사용하여 JSON 웹 API와를 구현 AWS SDK for Java 하여 Amazon DynamoDB에 데이터를 유지합니다. 애플리케이션의 서블릿 필터는 애플리케이션에서 처리하는 모든 수신 요청을 계측하고 SDK AWS 클라이언트의 요청 핸들러는 DynamoDB에 대한 다운스트림 호출을 계측합니다.

AWS Management Console 또는를 사용하여이 자습서를 따를 수 있습니다 AWS CLI.

Sections

사전 조건

이 자습서에서는 AWS CloudFormation 를 사용하여 샘플 애플리케이션 및 X-Ray 데몬을 실행하는 리소스를 생성하고 구성합니다. 튜토리얼을 설치하고 실행하려면 다음 사전 요구 사항이 필요합니다:

1. 권한이 제한된 IAM 사용자를 사용하는 경우 IAM 콘솔에서 다음 사용자 정책을 추가하십시오.
   - `AWSCloudFormationFullAccess`— CloudFormation 액세스 및 사용
   - `AmazonS3FullAccess` -를 사용하여 템플릿 파일을 CloudFormation에 업로드하려면 AWS Management Console
   - `IAMFullAccess`— Amazon ECS 및 Amazon EC2 인스턴스 역할 생성
   - `AmazonEC2FullAccess`— Amazon EC2 리소스 생성
   - `AmazonDynamoDBFullAccess`— DynamoDB 테이블 생성
   - `AmazonECS_FullAccess`— Amazon ECS 리소스 생성
   - `AmazonSNSFullAccess`— Amazon SNS 주제 생성
   - `AWSXrayReadOnlyAccess` — X-Ray 콘솔에서 트레이스 맵과 트레이스를 볼 수 있는 권한
2. 를 사용하여 자습서를 실행하려면 CLI 버전 2.7.9 이상을 AWS CLI설치하고 이전 단계의 사용자로 CLI를 구성합니다. https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html AWS CLI 사용자로를 구성할 때 리전이 구성되어 있는지 확인합니다. 지역이 구성되지 않은 경우 모든 CLI 명령에 `--region` *`AWS-REGION`*을 추가해야 합니다.
3. 샘플 애플리케이션 리포지토리를 복제하려면 Git이 설치되어 있어야 합니다.

4. 다음 코드 예제를 사용하여 Scorekeep 리포지토리의 `xray-gettingstarted` 브랜치를 복제합니다.

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

CloudFormation을 사용하여 Scorekeep 애플리케이션 설치

AWS Management Console

를 사용하여 샘플 애플리케이션 설치 AWS Management Console

1. CloudFormation 콘솔 열기
2. 스택 생성을 선택한 다음 드롭다운 메뉴에서 새 리소스 사용을 선택합니다.
3. 템플릿 지정 섹션에서 템플릿 파일 업로드를 선택합니다.
4. 파일 선택을 선택하고 git repo를 복제할 때 생성된 `xray-scorekeep/cloudformation` 폴더로 이동한 다음 `cf-resources.yaml` 파일을 선택합니다.
5. 다음을 선택하여 계속 진행합니다.
6. 스택 이름 입력란에 `scorekeep`을 입력하고 페이지 하단에서 다음을 선택하여 계속 진행합니다. 참고로 이 튜토리얼의 나머지 부분에서는 스택의 이름이 `scorekeep`으로 지정되었다고 가정합니다.
7. 스택 옵션 구성 페이지의 하단으로 스크롤하여 다음을 선택하여 계속 진행합니다.
8. 리뷰 페이지 하단으로 스크롤하여 CloudFormation이 사용자 지정 이름으로 IAM 리소스를 만들 수 있음을 승인하는 체크박스를 선택한 다음, 스택 생성을 선택합니다.
9. 이제 CloudFormation 스택이 생성되고 있습니다. 스택 상태는 약 5분 동안 `CREATE_IN_PROGRESS`로 유지되다가 `CREATE_COMPLETE`로 변경됩니다. 상태는 주기적으로 새로 고쳐지며, 페이지를 새로 고칠 수도 있습니다.

AWS CLI

를 사용하여 샘플 애플리케이션 설치 AWS CLI

1. 튜토리얼 앞부분에서 복제한 `xray-scorekeep` 리포지토리의 `cloudformation` 폴더로 이동합니다:

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

2. 다음 AWS CLI 명령을 입력하여 CloudFormation 스택을 생성합니다.

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

3. CloudFormation 스택 상태가 `CREATE_COMPLETE`이 될 때까지 기다리십시오. 약 5분이 소요됩니다. 다음 AWS CLI 명령을 사용하여 상태를 확인합니다.

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

데이터 추적 생성

애플리케이션에는 프런트 엔드 웹 앱이 포함되어 있습니다. 웹 앱을 사용하여 API 트래픽을 생성하고 트레이스 데이터를 X-Ray로 전송합니다. 먼저 AWS Management Console 또는 AWS CLI를 사용하여 웹 앱 URL을 검색합니다.

AWS Management Console

를 사용하여 애플리케이션 URL 찾기 AWS Management Console

1. CloudFormation 콘솔 열기
2. Stacks 목록에서 `scorekeep`을 선택합니다.
3. `scorekeep` 스택 페이지에서 출력 탭을 선택하고 `LoadBalancerUrl` URL 링크를 선택하여 웹 애플리케이션을 엽니다.

AWS CLI

를 사용하여 애플리케이션 URL 찾기 AWS CLI

1. 다음 명령어를 사용하여 웹 애플리케이션의 URL을 표시합니다:

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

2. 이 URL을 복사하여 브라우저에서 열면 Scorekeep 웹 애플리케이션이 표시됩니다.

웹 애플리케이션을 사용하여 트레이스 데이터를 생성하십시오.

1. [Create]를 선택하여 사용자 및 세션을 생성합니다.
2. [game name]을 입력하고, [Rules]를 [Tic Tac Toe]로 설정한 다음 [Create]를 선택하여 게임을 생성합니다.
3. [Play]를 선택하여 게임을 시작합니다.
4. 타일을 선택하여 동작을 하고 게임 상태를 변경합니다.

이들 각 단계에서 사용자, 세션, 게임, 동작 및 상태 데이터를 읽고 쓰기 위해 API에 대한 HTTP 요청과 DynamoDB에 대한 다운스트림 호출이 생성됩니다.

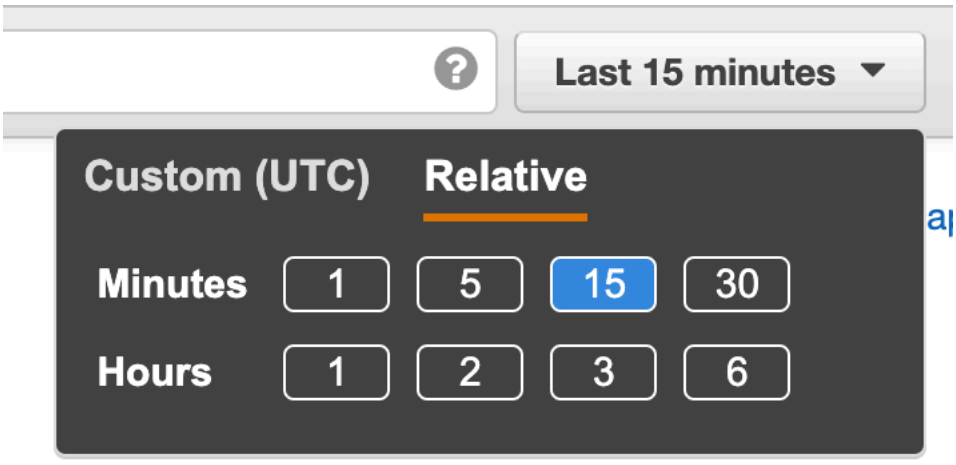
에서 트레이스 맵 보기 AWS Management Console

샘플 애플리케이션에서 생성된 트레이스 맵과 트레이스는 X-Ray 및 CloudWatch 콘솔에서 확인할 수 있습니다.

X-Ray console

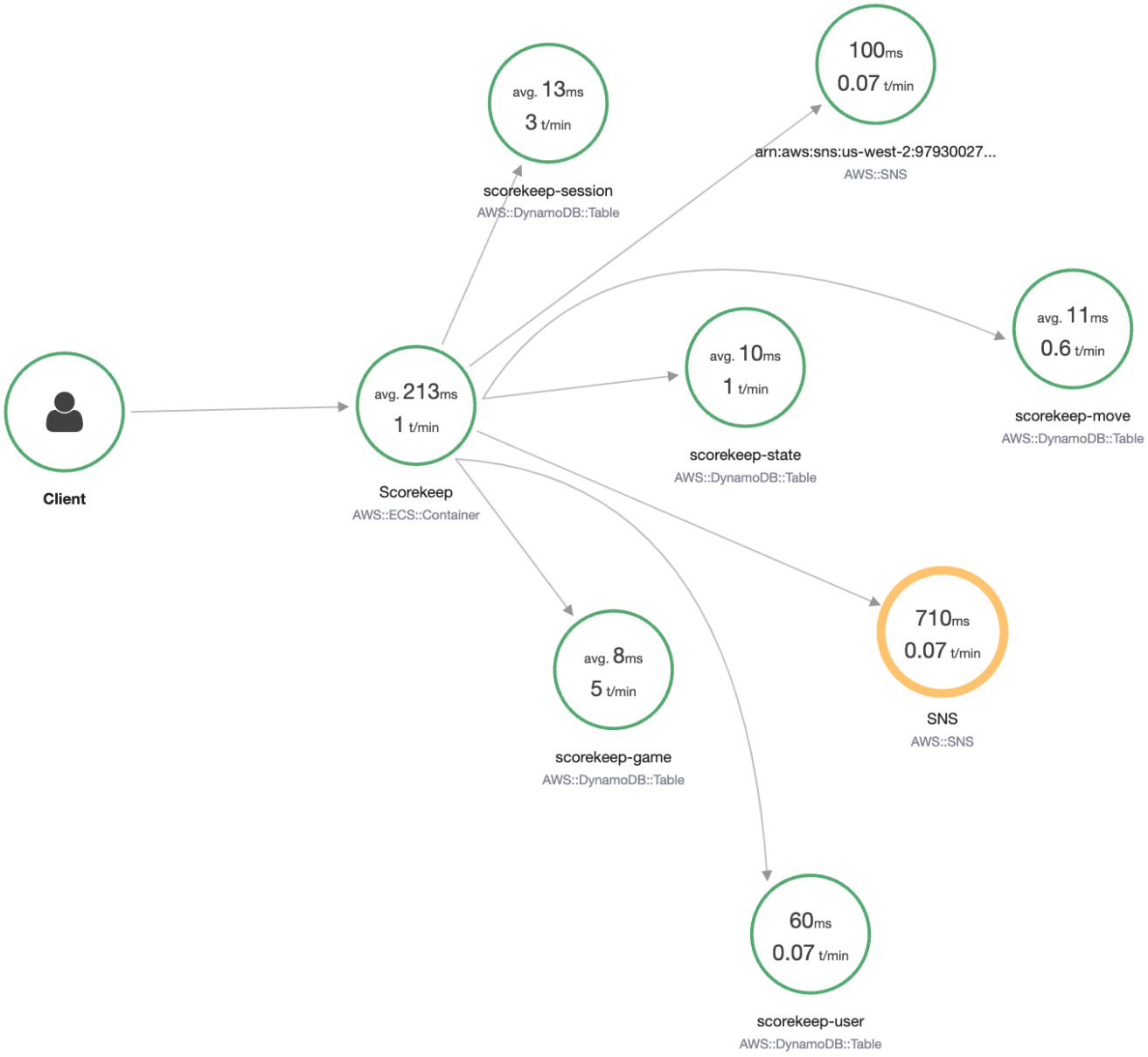
X-Ray 콘솔 사용하기

1. X-Ray 콘솔의 트레이스 맵 페이지를 엽니다.
2. 콘솔에 애플리케이션이 전송하는 트레이스 데이터로부터 X-Ray가 생성하는 서비스 그래프가 표시됩니다. 필요한 경우 트레이스 맵의 기간을 조정하여 웹 애플리케이션을 처음 시작한 이후 모든 트레이스가 표시되도록 합니다.



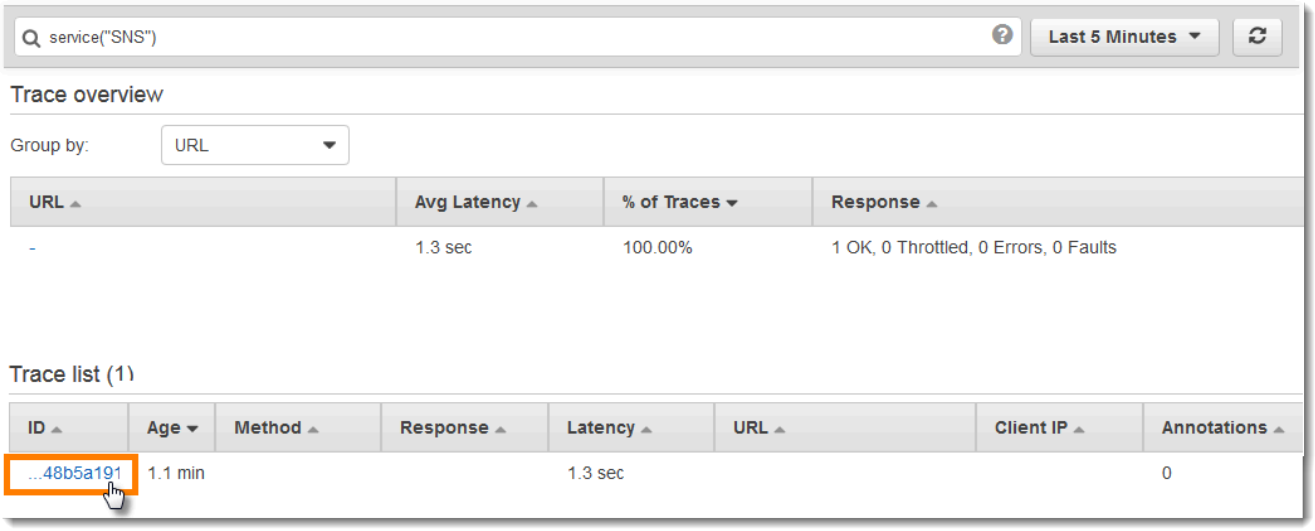
트레이스 맵에는 웹 앱 클라이언트, Amazon ECS에서 실행 중인 API, 애플리케이션에서 사용하는 각 DynamoDB 테이블이 표시됩니다. 애플리케이션에 대한 모든 요청에 대해, 요청이 API에 도달하고, 다운스트림 서비스를 요청하고, 완료될 때 매초 구성 가능한 최대 개수까지 요청이 트레이스됩니다.

서비스 그래프에서 노드를 선택하면 해당 노드로 트래픽을 생성한 요청의 트레이스를 볼 수 있습니다. 현재 Amazon SNS 노드는 노란색입니다. 드릴다운하여 이유를 확인하십시오.

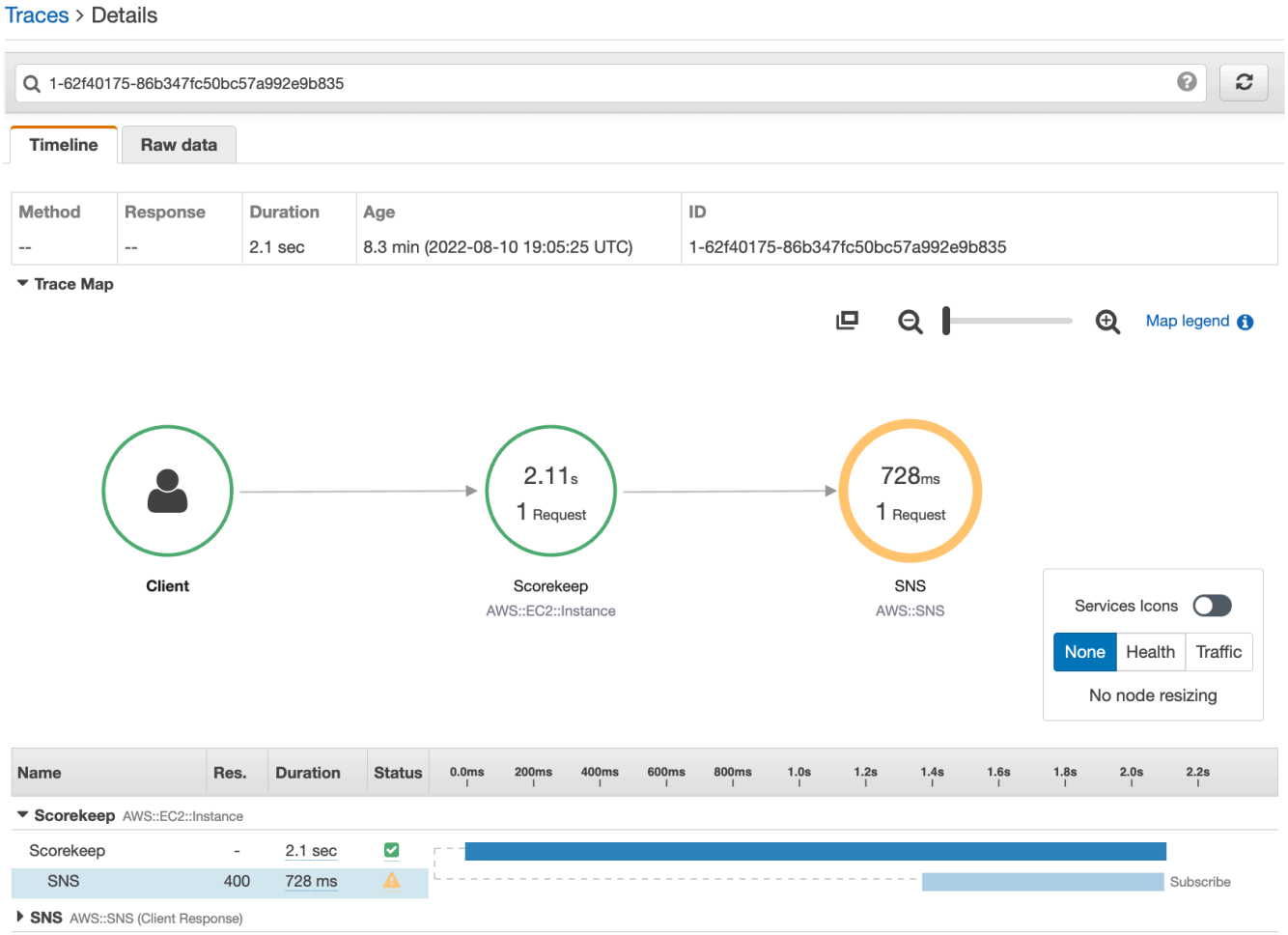


오류의 원인을 찾으려면

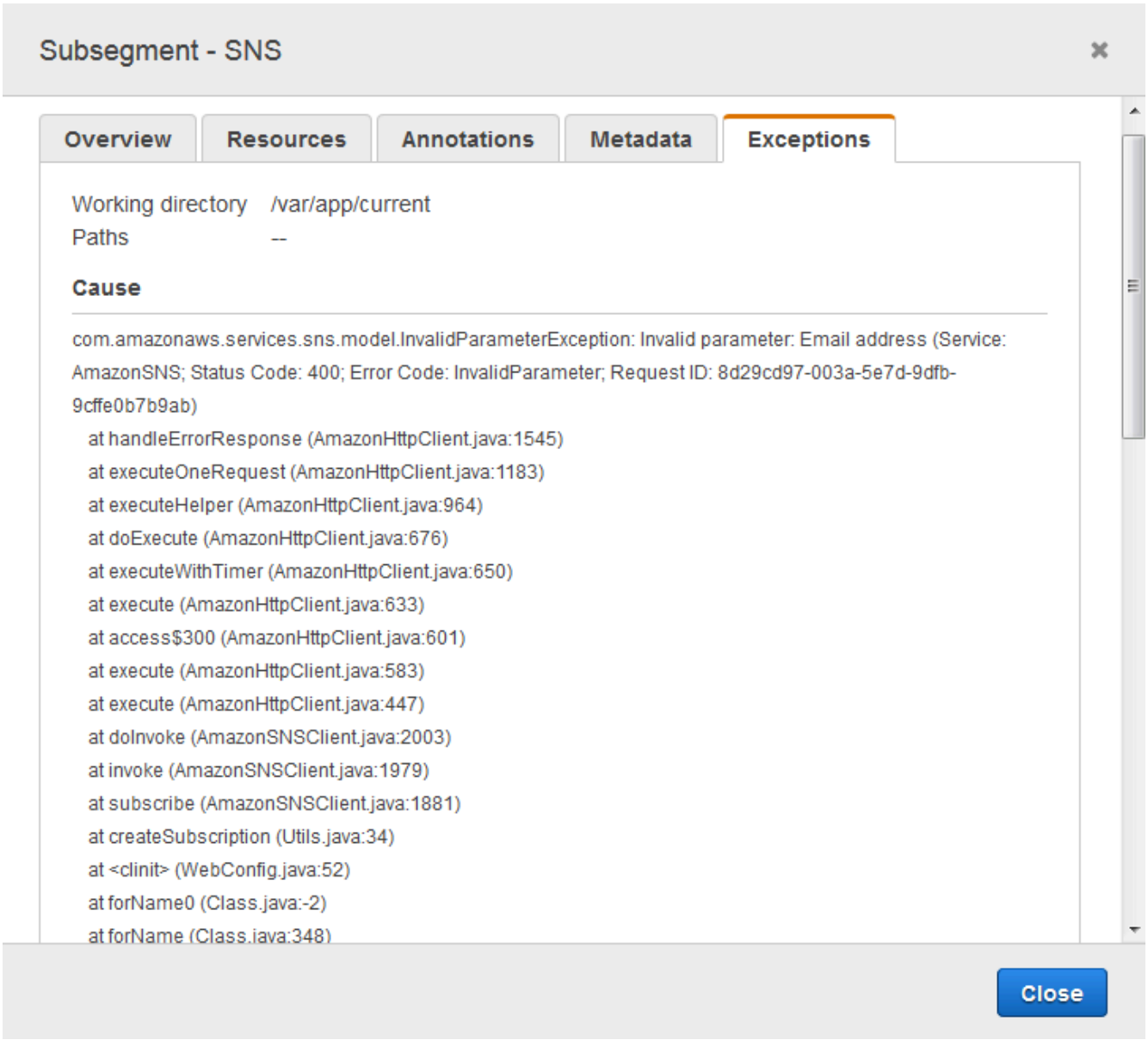
1. [SNS]라는 노드를 선택합니다. 노드 세부 정보 패널이 표시됩니다.
2. View traces(트레이스 보기)를 선택하여 Trace overview(트레이스 개요) 화면에 액세스합니다.
3. [Trace list]에서 트레이스를 선택합니다. 이 트레이스는 수신 요청에 대한 응답이 아니라 시작 시 기록된 것이므로 메서드 또는 URL이 없습니다.



4. 페이지 하단의 Amazon SNS 세그먼트에서 오류 상태 아이콘을 선택하여 SNS 하위 세그먼트의 예외 페이지가 열립니다.



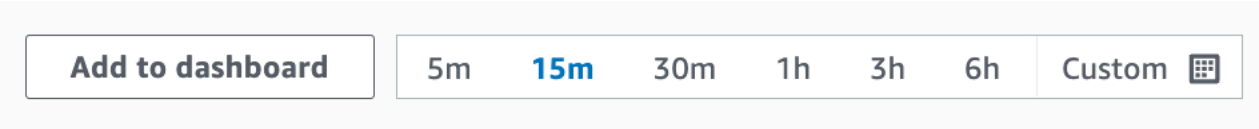
5. X-Ray SDK는 구성된 AWS SDK 클라이언트에서 발생하는 예외를 자동으로 캡처하고 스택 추적을 기록합니다.



CloudWatch console

CloudWatch 콘솔 사용하기

1. CloudWatch 콘솔의 X-Ray 트레이스 맵 페이지를 엽니다.
2. 콘솔에 애플리케이션이 전송하는 트레이스 데이터로부터 X-Ray가 생성하는 서비스 그래프가 표시됩니다. 필요한 경우 트레이스 맵의 기간을 조정하여 웹 애플리케이션을 처음 시작한 이후 모든 트레이스가 표시되도록 합니다.



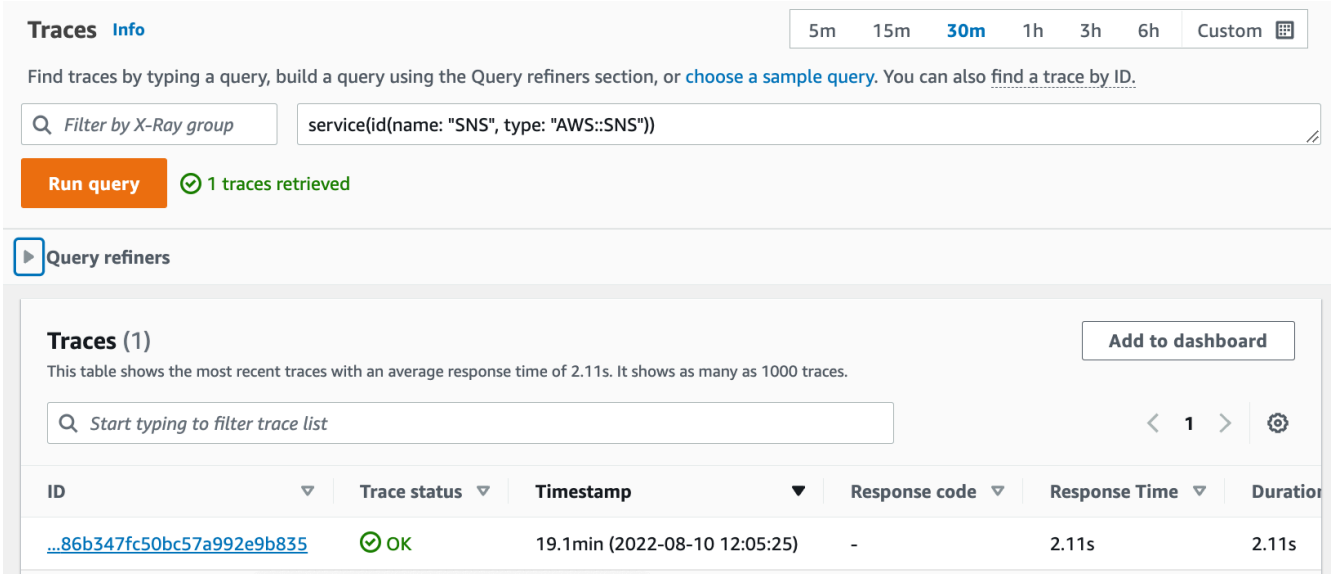
트레이스 맵에는 웹 앱 클라이언트, Amazon EC2에서 실행 중인 API, 애플리케이션에서 사용하는 각 DynamoDB 테이블이 표시됩니다. 애플리케이션에 대한 모든 요청에 대해, 요청이 API에 도달하고, 다운스트림 서비스를 요청하고, 완료될 때 매초 구성 가능한 최대 개수까지 요청이 트레이스됩니다.

서비스 그래프에서 노드를 선택하면 해당 노드로 트래픽을 생성한 요청의 트레이스를 볼 수 있습니다. 현재 Amazon SNS 노드는 주황색입니다. 드릴다운하여 이유를 확인하십시오.

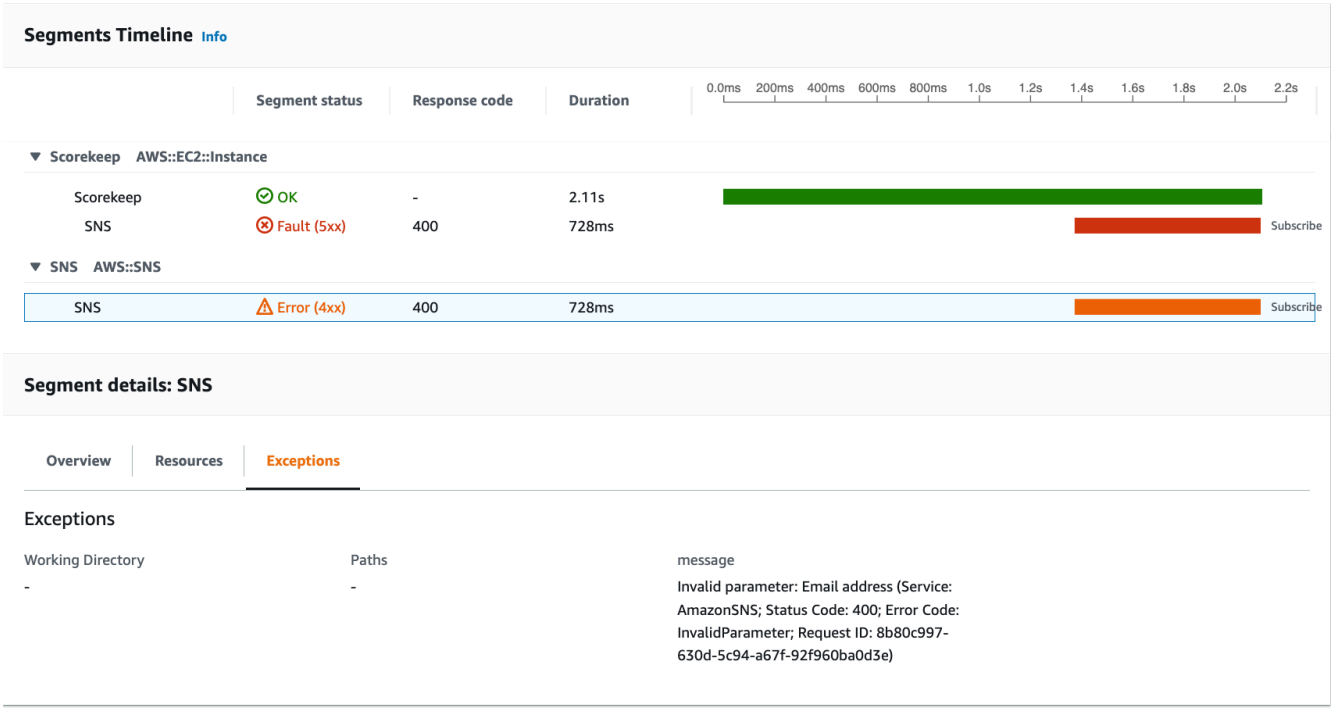

scorekeep-state
DynamoDB Table
scorekeep-game
DynamoDB Table
scorekeep-move
DynamoDB Table
Client
Scorekeep
ECS Container
SNS
SNS
scorekeep-session
DynamoDB Table

오류의 원인을 찾으려면

1. [SNS]라는 노드를 선택합니다. SNS 노드 세부 정보 패널은 지도 아래에 표시됩니다.
2. 추적 보기를 선택하여 추적 페이지에 액세스합니다.
3. 페이지 하단을 추가하고 추적 목록에서 추적을 선택합니다. 이 트레이스는 수신 요청에 대한 응답이 아니라 시작 시 기록된 것이므로 메서드 또는 URL이 없습니다.



4. 세그먼트 타임라인 하단에서 Amazon SNS 하위 세그먼트를 선택하고, SNS 하위 세그먼트의 예외 탭을 선택하여 예외 세부 정보를 확인합니다.



원인은 `WebConfig` 클래스에서 이루어진 `createSubscription` 직접 호출에서 제공된 이메일 주소가 잘못된 것입니다. 다음 섹션에서는 이 문제를 해결하겠습니다.

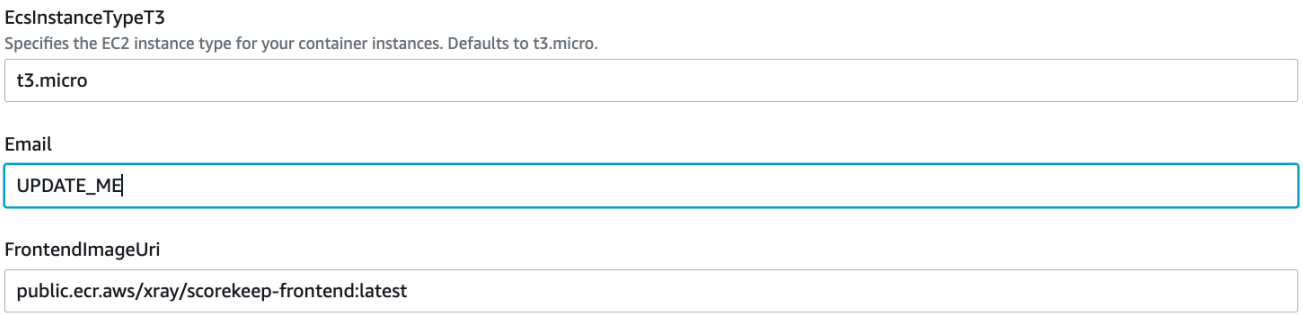
Amazon SNS 알림 구성

Scorekeep은 Amazon SNS를 사용하여 사용자가 게임을 완료하면 알림을 전송합니다. 애플리케이션이 시작되면 CloudFormation 스택 매개변수에 정의된 이메일 주소에 대한 구독을 생성하려고 시도합니다. 해당 직접 호출은 현재 실패하고 있습니다. 알림을 사용하도록 알림 이메일을 구성하고 트레이스 맵에 강조 표시된 오류를 해결합니다.

AWS Management Console

를 사용하여 Amazon SNS 알림을 구성하려면 AWS Management Console

1. CloudFormation 콘솔 열기
2. 목록에서 `scorekeep` 스택 이름 옆에 있는 라디오 버튼을 선택한 다음 업데이트를 선택합니다.
3. 현재 템플릿 사용이 선택되어 있는지 확인하고 업데이트 스택 페이지에서 다음을 클릭합니다.
4. 목록에서 이메일 매개 변수를 찾아 기본값을 유효한 이메일 주소로 바꿉니다.



5. 페이지의 하단으로 스크롤하고 다음(Next)을 선택합니다.
6. 리뷰 페이지 하단으로 스크롤하여 CloudFormation이 사용자 지정 이름으로 IAM 리소스를 만들 수 있음을 승인하는 체크박스를 선택한 다음, 스택 업데이트 선택합니다.
7. 이제 CloudFormation 스택이 업데이트되고 있습니다. 스택 상태는 약 5분 동안 `UPDATE_IN_PROGRESS`로 유지되다가 `UPDATE_COMPLETE`로 변경됩니다. 상태는 주기적으로 새로 고쳐지며, 페이지를 새로 고칠 수도 있습니다.

AWS CLI

를 사용하여 Amazon SNS 알림을 구성하려면 AWS CLI

1. 이전에 만든 `xray-scorekeep/cloudformation/` 폴더로 이동하여 텍스트 편집기에서 `cf-resources.yaml` 파일을 엽니다.
2. 이메일 매개 변수에서 `Default` 값을 찾아 *`UPDATE_ME`*에서 유효한 이메일 주소로 변경합니다.

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

3. `cloudformation` 폴더에서 다음 AWS CLI 명령을 사용하여 CloudFormation 스택을 업데이트합니다.

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

4. CloudFormation 스택 상태가 `UPDATE_COMPLETE`이 될 때까지 기다리십시오. 몇 분 정도 소요됩니다. 다음 AWS CLI 명령을 사용하여 상태를 확인합니다.

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

업데이트가 완료되면 Scorekeep이 다시 시작하고 SNS 주제에 대한 구독을 생성합니다. 이메일을 확인하여 게임을 완료할 때 업데이트를 보기 위한 구독을 확인합니다. 트레이스 맵을 열어 SNS 직접 호출이 더 이상 실패하지 않는지 확인합니다.

샘플 애플리케이션 탐색

샘플 애플리케이션은 Java용 X-Ray SDK를 사용하도록 구성된 Java용 HTTP 웹 API입니다. CloudFormation 템플릿을 사용하여 애플리케이션을 배포하면 ECS에서 Scorekeep을 실행하는 데 필요한 DynamoDB 테이블, Amazon ECS 클러스터 및 기타 서비스를 생성합니다. ECS용 작업 정의 파일은 CloudFormation을 통해 생성됩니다. 이 파일은 ECS 클러스터에서 작업별로 사용되는 컨테이너 이미지를 정의합니다. 이 이미지는 공식 X-Ray 공개 ECR에서 가져온 것입니다. Scorekeep API 컨테이너 이미지에는 Gradle로 컴파일된 API가 있습니다. Scorekeep 프론트엔드 컨테이너의 컨테이너 이

미지는 nginx 프록시 서버를 사용하여 프론트엔드에 서비스를 제공합니다. 이 서버는 /api로 시작하는 경로에 대한 요청을 API로 라우팅합니다.

수신 HTTP 요청을 구성하기 위해 애플리케이션은 SDK가 제공하는 `TracingFilter`를 추가합니다.

Example src/main/java/scorekeep/WebConfig.java – 서블릿 필터

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

@Configuration
public class WebConfig {

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

이 필터는 애플리케이션이 처리하는 모든 수신 요청에 대해 요청 URL, 메소드, 응답 상태, 시작 시간 및 종료 시간을 포함한 트레이스 데이터를 전송합니다.

또한 애플리케이션은 AWS SDK for Java를 사용하여 DynamoDB에 대한 다운스트림 호출을 생성합니다. 이러한 호출을 계측하기 위해 애플리케이션은 SDK AWS 관련 하위 모듈을 종속성으로 사용하고 Java용 X-Ray SDK는 모든 AWS SDK 클라이언트를 자동으로 계측합니다.

애플리케이션은 `Docker`을 사용하여 `Gradle Docker Image`와 함께 소스 코드를 인스턴스 내에서 빌드하고 `Scorekeep API Dockerfile` 파일을 사용하여 Gradle이 `ENTRYPOINT`에서 생성하는 실행 가능한 JAR을 실행합니다.

Example Gradle 도커 이미지를 통해 빌드하기 위해 도커 사용

```
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` 파일이 컴파일 도중 SDK 하위 모듈을 종속성으로 선언하여 Maven으로부터 이들 하위 모듈을 다운로드합니다.

Example build.gradle -- 종속성

```
...
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")
    }
}
```

코어, AWS SDK 및 AWS SDK Instrumentor 하위 모듈은 AWS SDK로 이루어진 다운스트림 호출을 자동으로 계측하는 데 필요한 모든 것입니다.

원시 세그먼트 데이터를 X-Ray API로 전달하려면 X-Ray 대몬(daemon)이 UDP 포트 2000에서 트래픽을 수신해야 합니다. 이를 위해 애플리케이션은 컨테이너에서 X-Ray 대몬(daemon)을 실행하고, 이 컨테이너는 사이드카 컨테이너로 ECS에 Scorekeep 애플리케이션과 함께 배포됩니다. 자세한 내용은 X-Ray 대몬(daemon) 항목을 참조하세요.

Example ECS 작업 정의의 X-Ray 대몬(daemon) 컨테이너 정의

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

Java용 X-Ray SDK는 코드를 계측하는 데 사용할 수 있는 TracingHandler인 전역 레코더를 제공하는 AWSXRay라는 클래스를 제공합니다. 전역 레코더를 구성하여 수신 HTTP 호출에 대해 세그먼트를 생성하는 AWSXRayServletFilter를 사용자 지정할 수 있습니다. 이 샘플은 WebConfig 클래스에 플러그인 및 샘플링 규칙을 사용하여 전역 레코더를 구성하는 정적 블록을 포함합니다.

Example src/main/java/scorekeep/WebConfig.java - 레코더

```
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());
    ...

  }
}
```

이 예제는 빌더를 사용하여 sampling-rules.json 파일로부터 샘플링 규칙을 로드합니다. 샘플링 규칙은 SDK가 들어오는 요청의 세그먼트를 기록하는 비율에 따라 다릅니다.

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
  }
}
```

샘플링 규칙 파일은 4개의 사용자 지정 샘플링 규칙 및 기본 규칙을 정의합니다. 각 수신 요청에 대해, SDK가 사용자 지정 규칙을 정의된 순서대로 평가합니다. SDK는 요청의 메서드, 경로 및 서비스 이름과 일치하는 첫 번째 규칙을 적용합니다. Scorekeep의 경우, 첫 번째 규칙이 초당 1개 요청의 고정 타겟, 그리고 고정 타겟 충족 이후 1.0 즉, 100%의 요청 비율을 적용하여 모든 POST 요청(리소스 생성 호출)을 포착합니다.

다른 세 개의 사용자 지정 규칙은 고정 타겟 없이 5% 비율을 세션, 게임 및 상태 읽기(GET 요청)에 적용합니다. 이는 콘텐츠가 최신 상태를 유지하도록 하기 위해 프런트 엔드가 몇 초마다 자동으로 생성하는 주기적 호출에 대한 트레이스 수를 최소화합니다. 다른 모든 요청의 경우 파일이 초당 1개의 기본 요청 비율과 10% 비율을 정의합니다.

또한 샘플 애플리케이션은 수동 SDK 클라이언트 구성, 추가 하위 세그먼트 생성, 발신 HTTP 호출과 같은 고급 기능을 사용하는 방법도 보여줍니다. 자세한 내용은 AWS X-Ray 샘플 애플리케이션 단원을 참조하세요.

선택 사항: 최소 권한 정책

Scorekeep ECS 컨테이너는 `AmazonSNSFullAccess` 및 `AmazonDynamoDBFullAccess`와 같은 전체 액세스 정책을 사용하여 리소스에 액세스합니다. 전체 액세스 정책을 사용하는 것은 프로덕션 애플리케이션의 모범 사례가 아닙니다. 다음 예제에서는 애플리케이션의 보안을 개선하기 위해 DynamoDB IAM 정책을 업데이트합니다. IAM 정책의 보안 모범 사례에 대한 자세한 내용은 AWS X-Ray용 자격 증명 및 액세스 관리를 참조하세요.

Example cf-resources.yaml 템플릿 ECSTaskRole 정의

```
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"
```

정책을 업데이트하려면 먼저 DynamoDB 리소스의 ARN을 식별합니다. 그런 다음 사용자 지정 IAM 정책에 ARN을 사용합니다. 마지막으로 이러한 정책을 인스턴스 프로파일에 적용합니다.

DynamoDB 리소스의 ARN을 식별하려면:

1. DynamoDB 콘솔을 엽니다.
2. 왼쪽 메뉴에서 테이블을 선택합니다.
3. `scorekeep-*` 중 하나를 선택하여 테이블 세부 정보 페이지를 표시합니다.
4. 개요 탭에서 추가 정보를 선택하여 섹션을 펼치고 Amazon 리소스 이름(ARN)을 확인합니다. 이 값을 복사합니다.
5. ARN을 다음 IAM 정책에 삽입하고 `AWS_REGION` 및 `AWS_ACCOUNT_ID` 값을 특정 지역 및 계정 ID로 변경합니다. 이 새 정책은 모든 작업을 허용하는 `AmazonDynamoDBFullAccess` 정책 대신 지정된 작업만 허용합니다.

   Example

   JSON

```
{
    "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-*"
        }
    ]
```

```
}
```

애플리케이션이 생성하는 테이블은 일관된 명명 규칙을 따릅니다. `scorekeep-*` 형식을 사용하여 모든 Scorekeep 테이블을 표시할 수 있습니다.

IAM 정책 변경하기

1. IAM 콘솔에서 Scorekeep 작업 역할(scorekeepRole)을 엽니다.
2. `AmazonDynamoDBFullAccess` 정책 옆에 있는 확인란을 선택하고 삭제 를 선택하여 이 정책을 삭제합니다.
3. 권한 추가를 선택한 다음 정책 연결을 선택하고 마지막으로 정책 생성을 선택합니다.
4. JSON 탭을 선택하고 위에서 생성한 정책을 붙여넣습니다.
5. 페이지 하단에서 다음: 태그를 선택합니다.
6. 페이지 하단에서 다음: 리뷰를 선택합니다.
7. 이름(Name)에 정책 이름을 입력합니다.
8. 페이지 하단에서 정책 생성을 선택합니다.
9. `scorekeepRole` 역할에 새로 생성한 정책을 연결합니다. 첨부된 정책이 적용되려면 몇 분 정도 소요될 수 있습니다.

새 정책을 `scorekeepRole` 역할에 연결한 경우 CloudFormation 스택을 삭제하기 전에 새 정책을 분리해야 합니다. 이렇게 연결된 정책은 스택 삭제를 차단하기 때문입니다. 정책을 삭제하면 정책이 자동으로 분리될 수 있습니다.

사용자 지정 IAM 정책 제거

1. IAM 콘솔을 엽니다.
2. 왼쪽 탐색 메뉴에서 정책을 선택합니다.
3. 이 섹션의 앞부분에서 생성한 사용자 지정 정책 이름을 검색하고 정책 이름 옆의 라디오 버튼을 선택하여 강조 표시합니다.
4. 작업 드롭다운을 선택한 다음 삭제를 선택합니다.
5. 사용자 지정 정책의 이름을 입력한 다음 삭제를 선택하여 삭제합니다. 그러면 정책이 `scorekeepRole` 역할에서 자동으로 분리됩니다.

정리

Scorekeep 애플리케이션 리소스를 삭제하려면 다음 절차를 따르세요.

> **Note**
>
> 이 튜토리얼의 이전 섹션을 사용하여 사용자 지정 정책을 생성하고 연결한 경우 CloudFormation 스택을 삭제하기 전에 `scorekeepRole`에서 정책을 제거해야 합니다.

AWS Management Console

를 사용하여 샘플 애플리케이션 삭제 AWS Management Console

1. CloudFormation 콘솔 열기
2. 목록에서 `scorekeep` 스택 이름 옆에 있는 라디오 버튼을 선택한 다음 삭제를 선택합니다.
3. 이제 CloudFormation 스택이 삭제됩니다. 스택 상태는 모든 리소스가 삭제될 때까지 몇 분 동안 `DELETE_IN_PROGRESS`로 유지됩니다. 상태는 주기적으로 새로 고쳐지며, 페이지를 새로 고칠 수도 있습니다.

AWS CLI

를 사용하여 샘플 애플리케이션 삭제 AWS CLI

1. 다음 AWS CLI 명령을 입력하여 CloudFormation 스택을 삭제합니다.

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

2. CloudFormation 스택이 더 이상 존재하지 않을 때까지 기다리십시오. 약 5분 정도 소요됩니다. 다음 AWS CLI 명령을 사용하여 상태를 확인합니다.

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

다음 단계

다음 장 AWS X-Ray 개념에서 X-Ray에 대해 자세히 알아봅니다.

자체 앱을 계측하려면 Java용 X-Ray SDK 또는 다른 X-Ray SDK 중 하나에 대해 자세히 알아보십시오.

- Java용 X-Ray SDK – AWS X-Ray Java용 SDK
- Node.js용 X-Ray SDK – AWS Node.js용 X-Ray SDK
- .NET용 X-Ray SDK – AWS X-Ray .NET용 SDK

로컬 또는에서 X-Ray 데몬을 실행하려면 섹션을 AWS참조하세요AWS X-Ray 데몬.

GitHub에서 샘플 애플리케이션에 기여하려면 eb-java-scorekeep 단원을 참조하십시오.

AWS SDK 클라이언트 수동 계측

Java용 X-Ray SDK는 빌드 종속성에 AWS SDK Instrumentor 하위 모듈을 포함하면 모든 SDK 클라이언트를 자동으로 계측합니다. AWS

Instrumentor 하위 모듈을 제거하여 자동 클라이언트 구성을 비활성화할 수 있습니다. 그러면 다른 클라이언트는 무시하고 일부 클라이언트만 수동으로 구성하거나 클라이언트마다 다른 트레이스 핸들러를 사용할 수 있습니다.

특정 AWS SDK 클라이언트 계측 지원을 설명하기 위해 애플리케이션은 추적 핸들러를 사용자, 게임 및 세션 모델의 요청 핸들러AmazonDynamoDBClientBuilder로에 전달합니다. 이 코드 변경은 SDK가 해당 클라이언트를 사용하여 DynamoDB에 대한 호출을 모두 계측하도록 지시합니다.

Example **src/main/java/scorekeep/SessionModel.java** – 수동 AWS SDK 클라이언트 구성

```
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);
```

프로젝트 종속성에서 AWS SDK Instrumentor 하위 모듈을 제거하면 수동으로 구성된 AWS SDK 클라이언트만 트레이스 맵에 나타납니다.

추가 하위 세그먼트 생성

사용자 모델 클래스에서 애플리케이션은 수동으로 하위 세그먼트를 생성하여 saveUser 함수에서 생성된 모든 다운스트림 호출을 그룹화하고 메타데이터를 추가합니다.

Example **src/main/java/scorekeep/UserModel.java** - 사용자 지정 하위 세그먼트

```
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();
    }
  }
```

주석, 메타데이터 및 사용자 ID 기록

게임 모델 클래스에서 애플리케이션은 게임을 DynamoDB에 저장할 때마다 Game 객체를 메타데이터 블록에 기록합니다. 이와 별도로, 애플리케이션은 필터 표현식에 사용할 수 있도록 게임 ID를 주석 에 기록합니다.

Example **src/main/java/scorekeep/GameModel.java** – 주석 및 메타데이터

```
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();
    }
  }
```

이동 컨트롤러의 경우, 애플리케이션은 사용자 ID를 setUser와 함께 기록합니다. 사용자 ID는 세그먼트의 별도 필드에 기록되면 검색용으로 인덱스되지 않습니다.

Example src/main/java/scorekeep/MoveController.java – 사용자 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);
  }
```

발신 HTTP 호출 구성

사용자 팩토리 클래스는 애플리케이션이 Java용 X-Ray SDK의 HTTPClientBuilder를 사용하여 발신 HTTP 직접 호출을 계측하는 방법을 보여줍니다.

Example **src/main/java/scorekeep/UserFactory.java** – HTTPClient 구성

```
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();
    }
  }
```

현재 `org.apache.http.impl.client.HttpClientBuilder`를 사용 중인 경우 해당 클래스에 대한 import 문을 `com.amazonaws.xray.proxies.apache.http.HttpClientBuilder`에 대한 문으로 바꿀 수 있습니다.

PostgreSQL 데이터베이스에 대한 호출 구성

`application-pgsql.properties` 파일은 `RdsWebConfig.java`에서 생성된 데이터 소스에 X-Ray PostgreSQL 추적 인터셉터를 추가합니다.

Example **`application-pgsql.properties`** – PostgreSQL 데이터베이스 구성

```
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**
>
> PostgreSQL 데이터베이스를 애플리케이션 환경에 추가하는 방법에 관한 자세한 정보는 AWS Elastic Beanstalk 개발자 안내서의 Elastic Beanstalk로 데이터베이스 구성을 참조하십시오.

`xray` 브랜치의 X-Ray 데모 페이지에는 계측된 데이터 소스를 사용하여 생성한 SQL 쿼리에 대한 정보를 표시하는 트레이스를 생성하는 데모가 포함되어 있습니다. 데모 페이지를 보려면 실행 중인 애플리케이션의 `/#/xray` 경로로 이동하거나 탐색 모음에서 Powered by AWS X-Ray를 선택하십시오.

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.

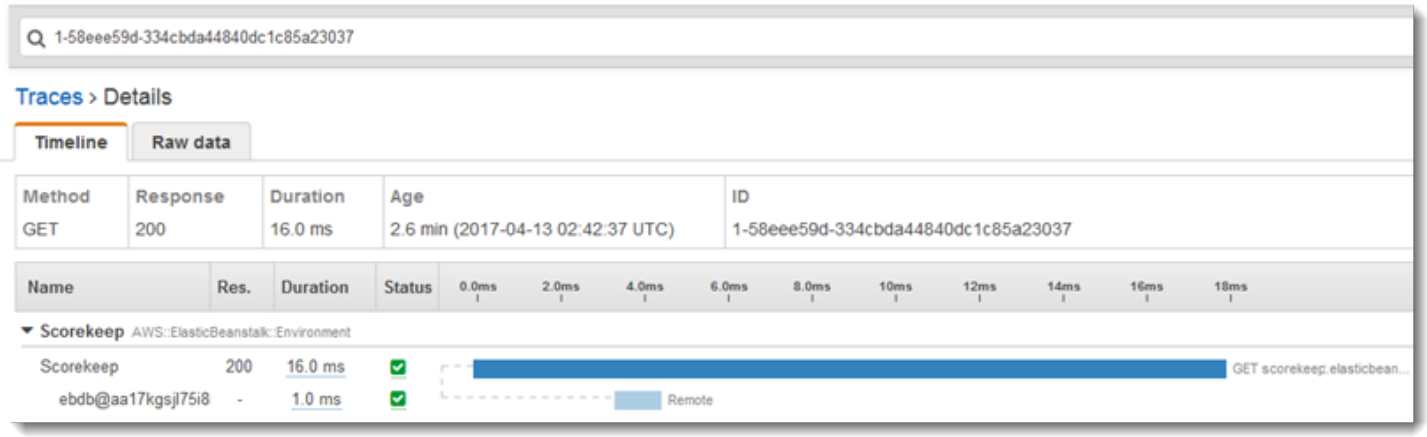
Trace SQL queries

View traces in AWS X-Ray

ID	Winner	Loser
1	Mugur	Gheorghiță
2	Paula	Adorján
3	Αρχίας	Stela
4	付	Pərvanə

게임 세션을 시뮬레이션하고 연결된 데이터베이스에 결과를 저장하려면 [Trace SQL queries]를 선택합니다. 그런 다음 AWS X-Ray에서 트레이스 보기를 선택하여 API의 `/api/history` 경로에 도달하는 필터링된 트레이스 목록을 확인합니다.

목록에서 트레이스 하나를 선택해 SQL 쿼리를 포함한 타임라인을 확인하십시오.



AWS Lambda 함수 계측

Scorekeep은 두 가지 AWS Lambda 함수를 사용합니다. 첫 번째 함수는 새 사용자를 위한 임의의 이름을 생성하는 `lambda` 분기의 Node.js 함수입니다. 사용자가 이름을 입력하지 않고 세션을 생성하면 애플리케이션은 AWS SDK for Java를 사용하여 `random-name`이라는 함수를 호출합니다. Java용 X-Ray SDK는 계측된 AWS SDK 클라이언트로 이루어진 다른 호출과 마찬가지로 Lambda 호출에 대한 정보를 하위 세그먼트에 기록합니다.

> **Note**
>
> `random-name` Lambda 함수를 실행하려면 Elastic Beanstalk 환경 외부에서 추가 리소스를 생성해야 합니다. 자세한 내용과 지침은 추가 정보 파일(AWS Lambda Integration)을 참조하십시오.

두 번째 함수인 `scorekeep-worker`는 Scorekeep API와 독립적으로 실행하는 Python 함수입니다. 게임이 끝나면 이 API는 세션 ID와 게임 ID를 SQS 대기열에 씁니다. 작업자 함수는 대기열에서 항목을 읽고 Scorekeep API를 직접 호출하여 Amazon S3에 저장할 각 게임 세션의 전체 레코드를 생성합니다.

Scorekeep에는 두 함수를 모두 생성하기 위한 AWS CloudFormation 템플릿과 스크립트가 포함되어 있습니다. X-Ray SDK를 함수 코드와 함께 번들링해야 하므로 템플릿은 코드가 없는 함수를 생성합니

다. Scorekeep를 배포할 때 `.ebextensions` 폴더에 포함된 구성 파일은 SDK가 포함된 소스 번들을 생성하고 AWS Command Line Interface를 사용하여 함수 코드와 구성을 업데이트합니다.

함수

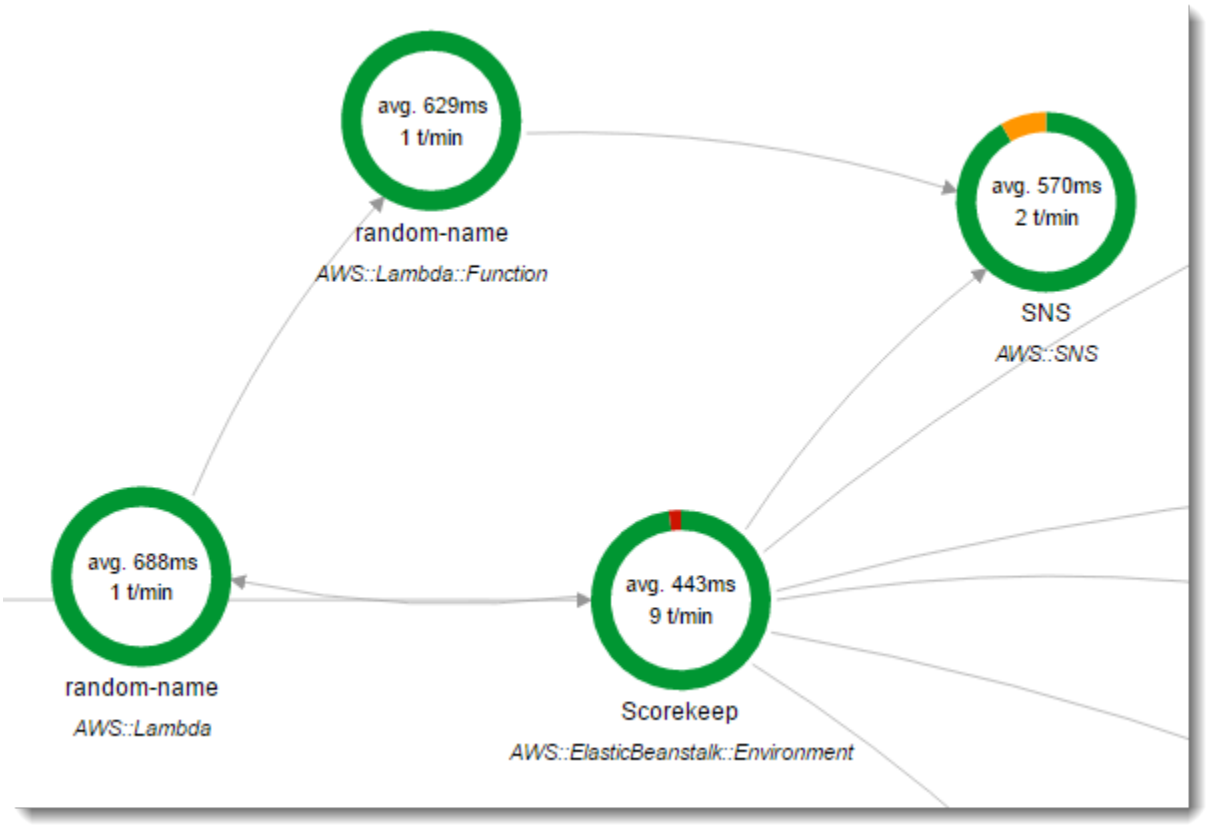
- Random name
- 작업자

Random name

사용자가 로그인하지 않거나 사용자 이름을 입력하지 않고 게임 세션을 시작하면 Scorekeep는 random name 함수를 호출합니다. Lambda는 `random-name`에 대한 직접 호출을 처리할 때 추적 헤더를 읽습니다. 이 헤더에는 Java용 X-Ray SDK가 기록한 추적 ID 및 샘플링 결정이 포함되어 있습니다.

샘플링된 각 요청에 대해 Lambda는 X-Ray 대몬(daemon)을 실행하고 두 개의 세그먼트를 기록합니다. 첫 번째 세그먼트에는 함수를 불러오는 Lambda 직접 호출에 대한 정보가 기록됩니다. 이 세그먼트에는 Scorekeep이 기록하는 하위 세그먼트와 동일한 정보가 포함되지만 Lambda 관점에서 기록됩니다. 두 번째 세그먼트에는 함수가 수행하는 작업이 표현됩니다.

Lambda는 함수 컨텍스트를 통해 함수 세그먼트를 X-Ray SDK에 전달합니다. Lambda 함수를 계측할 때는 SDK를 사용하여 수신 요청에 대한 세그먼트를 생성하지 않습니다. Lambda는 세그먼트를 제공하고, 사용자는 SDK를 사용하여 클라이언트를 계측하고 하위 세그먼트를 작성합니다.



`random-name` 함수는 Node.js에서 구현됩니다. Node.js의 JavaScript용 SDK를 사용하여 Amazon SNS로 알림을 보내고 Node.js용 X-Ray SDK를 사용하여 AWS SDK 클라이언트를 계측합니다. 주석을 쓰기 위해 이 함수는 `AWSXRay.captureFunc`를 사용하여 사용자 지정 하위 세그먼트를 만들고 구성된 함수에 주석을 씁니다. Lambda에서는 함수 세그먼트에 주석을 직접 쓸 수 없고 생성한 하위 세그먼트에만 쓸 수 있습니다.

Example **`function/index.js`** -- Random name Lambda 함수

```
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;
```

이 함수는 샘플 애플리케이션을 Elastic Beanstalk에 배포할 때 자동으로 생성됩니다. `xray` 브랜치에는 빈 Lambda 함수를 생성하기 위한 스크립트가 포함됩니다. `.ebextensions` 폴더의 구성 파일은 배포 `npm install` 중에 로 함수 패키지를 빌드한 다음 AWS CLI로 Lambda 함수를 업데이트합니다.

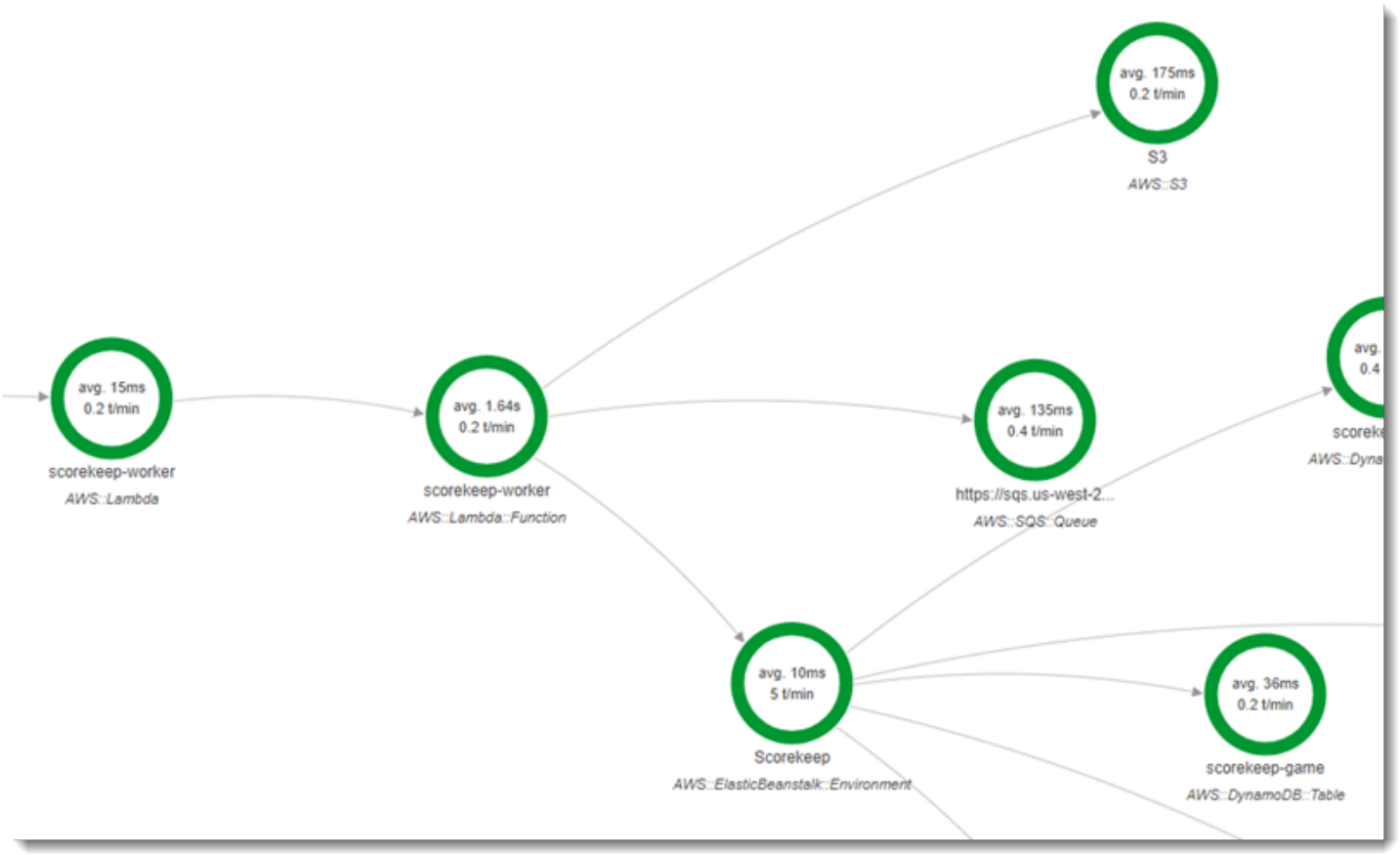
작업자

구성된 작업자 함수는 `xray-worker`라는 고유 분기에 제공됩니다. 작업자 함수와 관련 리소스를 먼저 생성하지 않은 상태에서는 실행할 수 없기 때문입니다. 지침은 분기 readme를 참조하십시오.

이 기능은 5분마다 번들로 제공되는 Amazon CloudWatch 이벤트에 의해 트리거됩니다. 이 함수가 실행되면 Scorekeep이 관리하는 Amazon SQS 대기열에서 항목을 가져옵니다. 각 메시지에는 완료된 게임에 대한 정보가 들어있습니다.

작업자는 게임 레코드와 게임 레코드가 참조하는 다른 테이블의 게임 문서를 가져옵니다. 예를 들면 DynamoDB의 게임 레코드에는 게임 중 실행된 동작 목록이 포함됩니다. 이 목록에는 이동 자체는 들어 있지 않고, 별도 테이블에 저장된 이동의 ID만 들어 있습니다.

세션과 상태도 참조로 저장됩니다. 따라서 게임 테이블의 항목이 너무 크지 않은 상태로 유지되지만 게임에 대한 모든 정보를 가져오려면 추가 호출이 필요합니다. 작업자는 이러한 모든 항목을 역참조하여 게임의 전체 레코드를 Amazon S3에서 단일 문서로 구성합니다. 데이터에 대한 분석을 수행하려면 읽기 중심의 데이터 마이그레이션을 실행하지 않고 Amazon Athena를 통해 Amazon S3에서 직접 쿼리를 실행하여 DynamoDB에서 데이터를 가져올 수 있습니다.



이 작업자 함수는 AWS Lambda의 구성에서 활성 추적이 활성화되어 있습니다. 무작위 이름 함수와 달리 작업자는 계측된 애플리케이션으로부터 요청을 수신하지 않으므로 AWS Lambda 는 추적 헤더를 수신하지 않습니다. Lambda는 활성 추적을 사용하여 추적 ID를 생성하고 샘플링 결정을 내립니다.

Python용 X-Ray SDK는 함수 상단에서 SDK를 가져오고 `patch_all` 함수를 실행하여 Amazon SQS 및 Amazon S3를 호출하는 데 사용하는 AWS SDK for Python (Boto) 및 HTTclients하는 몇 줄에 불과합니다. 이 작업자가 Scorekeep API를 호출하면 SDK는 추적 헤더를 요청에 추가하여 API를 통해 호출을 추적합니다.

Example **`_lambda/scorekeep-worker/scorekeep-worker.py`** -- 작업자 Lambda 함수

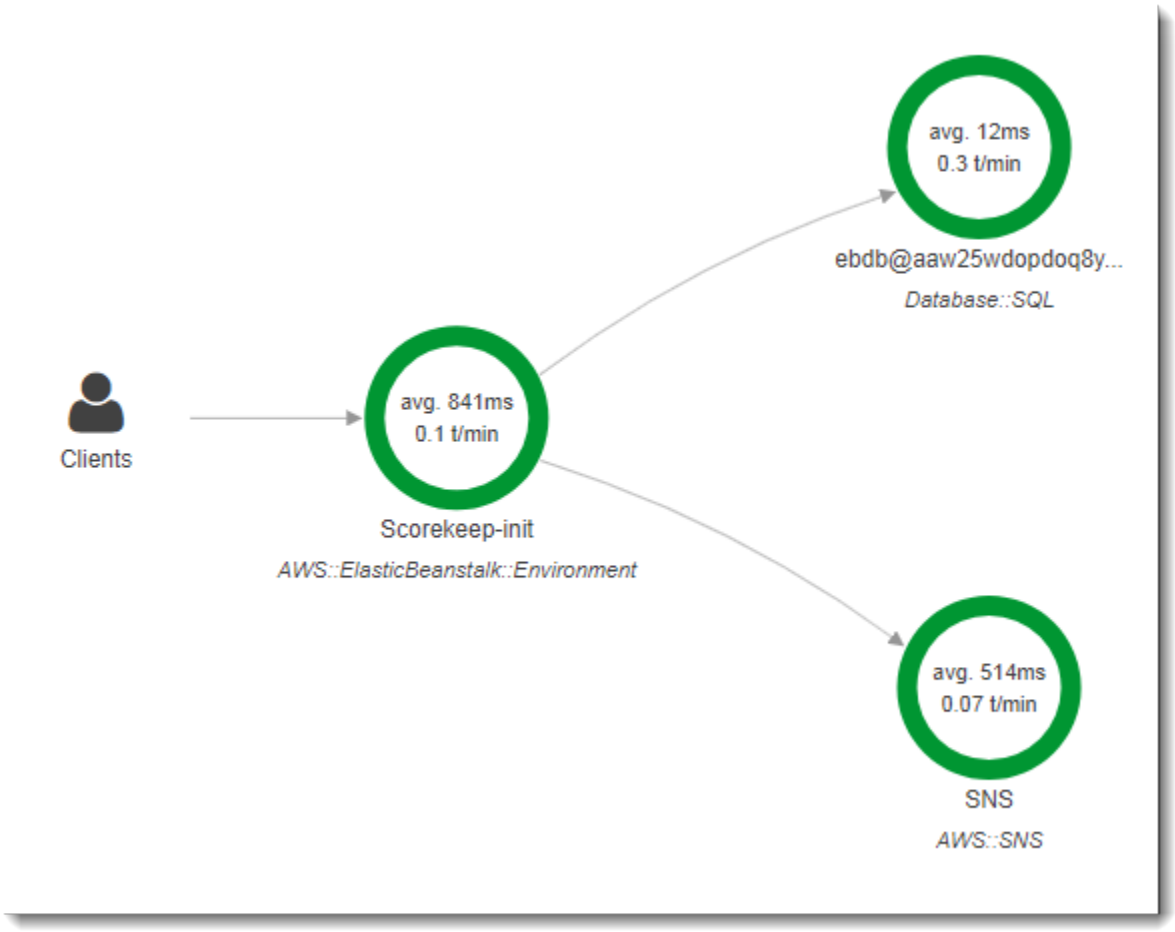
```
import os
import boto3
```

```
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
    )
   ...
```

시작 코드 구성

Java용 X-Ray DSK가 수신 요청에 대한 세그먼트를 자동으로 생성합니다. 요청이 범위 내에 있을 때에는 문제 없이 구성된 클라이언트를 사용하고 하위 세그먼트를 레코딩할 수 있습니다. 계측된 클라이언트를 시작 코드에서 사용하려는 경우 SegmentNotFoundException이 발생합니다.

시작 코드는 웹 애플리케이션의 표준 요청/응답 흐름 외부에서 실행되므로 시작 코드를 구성하려면 세그먼트를 수동으로 생성해야 합니다. Scorekeep이 해당 `WebConfig` 파일에서 시작 코드의 구성을 표시합니다. Scorekeep이 시작 중에 SQL 데이터베이스와 Amazon SNS를 호출합니다.



기본 `WebConfig` 클래스가 알림을 위한 Amazon SNS 구독을 생성합니다. Amazon SNS 클라이언트가 사용될 때 X-Ray SDK가 기록할 세그먼트를 제공하기 위해 Scorekeep이 전역 레코더에서 `beginSegment` 및 `endSegment`를 호출합니다.

Example **src/main/java/scorekeep/WebConfig.java** — 시작 코드에 계측된 AWS SDK 클라이언트

```
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();
```

Amazon RDS 데이터베이스가 연결되어 있을 때 Scorekeep이 사용하는 RdsWebConfig에서는 구성이 시작 중에 데이터베이스 스키마를 적용할 때 Hibernate가 사용하는 SQL 클라이언트에 대한 세그먼트도 생성합니다.

Example **src/main/java/scorekeep/RdsWebConfig.java** — 시작 코드에 계측된 SQL 데이터베이스 클라이언트

```
@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();
}
```

SchemaExport가 자동으로 실행되고 SQL 클라이언트가 사용됩니다. 클라언트가 구성되었으므로 Scorekeep이 기본 구현을 재정의하고 클라이언트가 호출될 때 SDK에서 사용할 세그먼트를 제공해야 합니다.

스크립트 구성

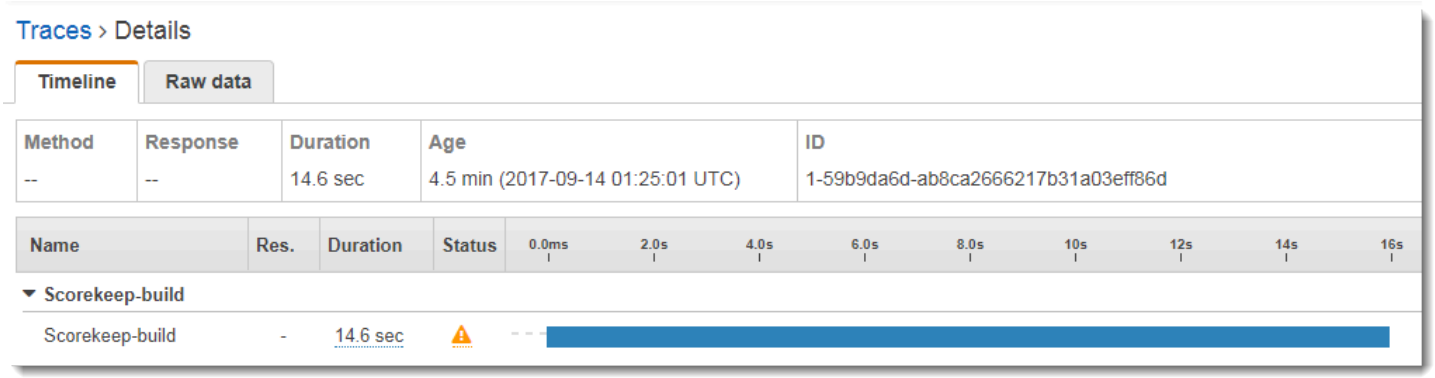
애플리케이션의 일부가 아닌 코드를 구성할 수도 있습니다. X-Ray 대몬(daemon)이 실행 중이면 X-Ray SDK에서 생성되지 않은 경우에도 수신하는 모든 세그먼트를 X=Ray로 릴레이합니다. Scorekeep은 해당 스크립트를 사용하여 배포하는 동안 애플리케이션을 컴파일하는 빌드를 구성합니다.

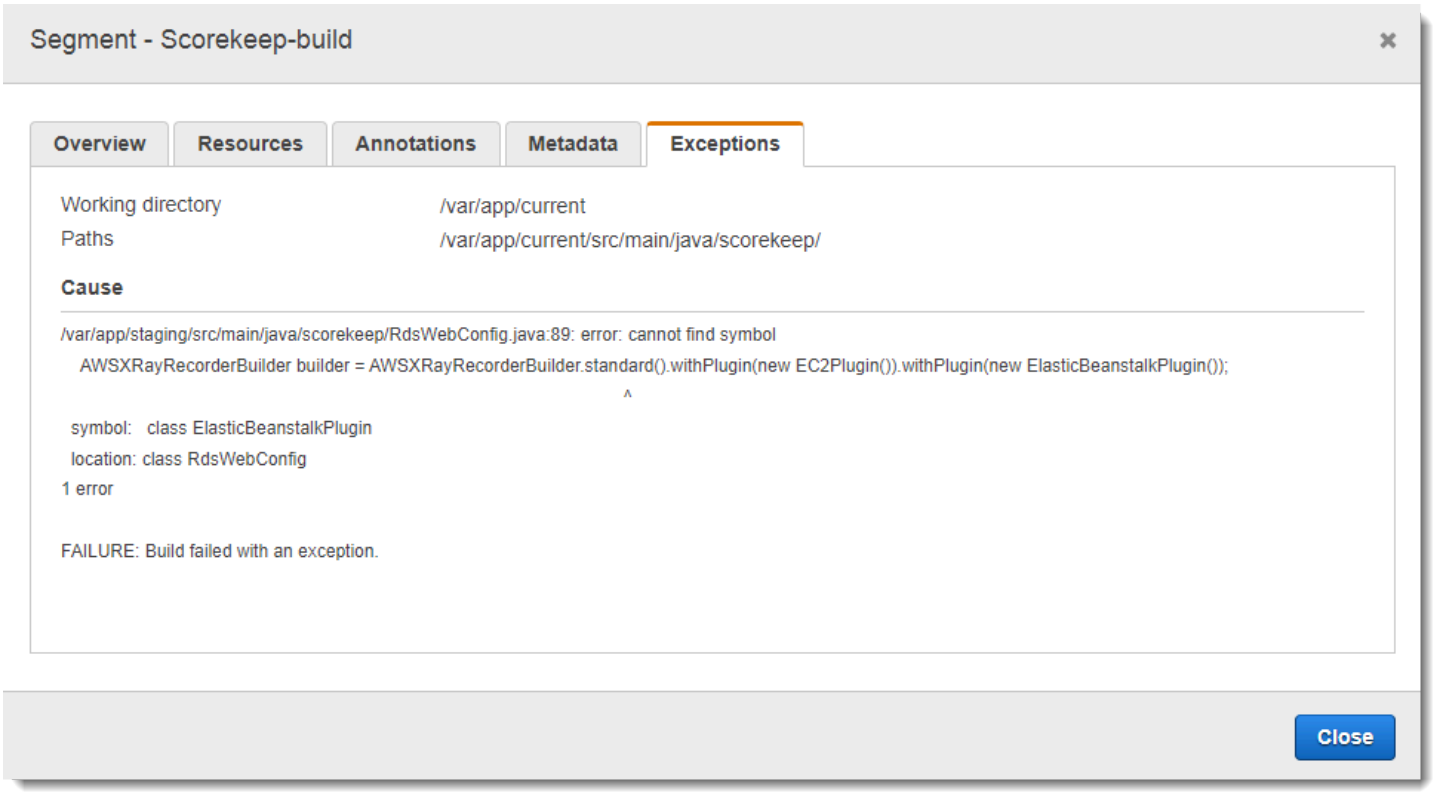
Example **bin/build.sh**— 계측된 빌드 스크립트

```
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 및 xray_success.py는 세그먼트 객체를 생성한 후 JSON 문서로 변환하고 UDP를 통해 데몬에 보내는 간단한 Python 스크립트입니다. Gradle 빌드가 실패하는 경우 X-Ray 콘솔 트레이스 맵에서 scorekeep-build 노드를 클릭하면 오류 메시지를 확인할 수 있습니다.





웹 앱 클라이언트 구성

xray-cognito 브랜치에서는 Scorekeep이 Amazon Cognito를 사용하므로 사용자가 계정을 생성하고 해당 계정으로 로그인하여 Amazon Cognito 사용자 풀에서 자신의 사용자 정보를 검색할 수 있습니다. 사용자가 로그인하면 Scorekeep은 Amazon Cognito 자격 증명 풀을 사용하여에 사용할 임시 AWS 자격 증명을 가져옵니다 AWS SDK for JavaScript.

자격 증명 풀은 로그인한 사용자가 추적 데이터를 AWS X-Ray에 쓸 수 있도록 구성되어 있습니다. 웹 앱은 이러한 자격 증명을 사용하여 로그인한 사용자의 ID, 브라우저 경로 및 Scorekeep API 호출에 대한 클라이언트의 보기를 레코딩합니다.

이러한 작업의 대부분은 이름이 `xray`인 서비스 클래스에서 수행됩니다. 이 서비스 클래스는 필수 식별자 생성, 진행 중인 세그먼트 생성, 세그먼트 종료, X-Ray API에 세그먼트 문서 전송 등의 작업을 위한 메서드를 제공합니다.

Example **public/xray.js**— 세그먼트 기록 및 업로드

```
...
  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);
    }
  })
```

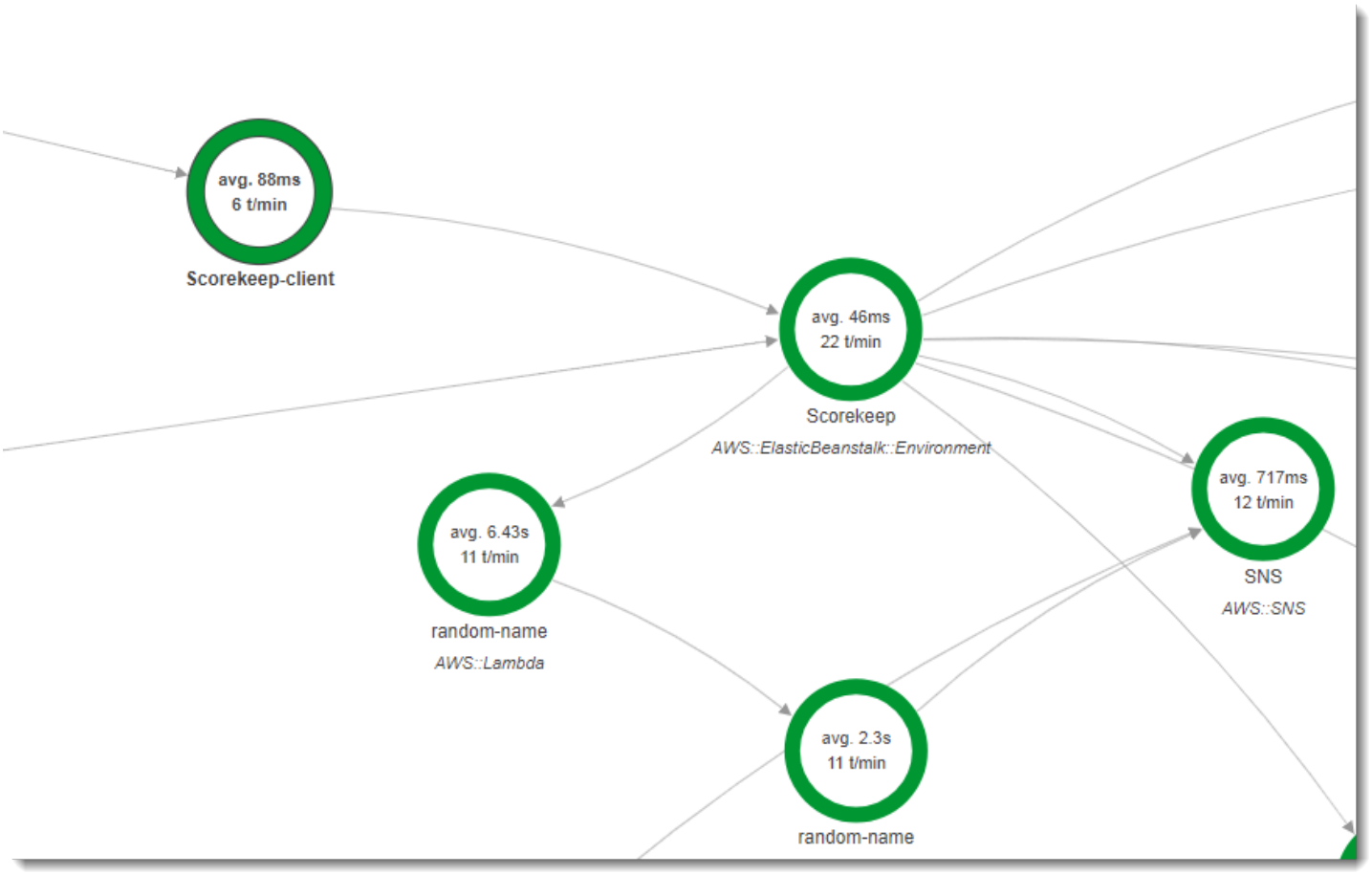
```
  }
```

이러한 메서드는 웹 앱이 Scorekeep API를 호출하는 데 사용하는 리소스 서비스의 헤더 및 `transformResponse` 함수에서 호출됩니다. API가 생성하는 세그먼트와 동일한 추적에 클라이언트 세그먼트를 포함하려면 웹 앱이 추적 헤더(`X-Amzn-Trace-Id`)에 추적 ID와 세그먼트 ID를 포함시켜야 하며, 이 헤더는 X-Ray SDK가 읽을 수 있어야 합니다. 구성된 Java 애플리케이션이 이 헤더와 함께 요청을 수신하면 Java용 X-Ray SDK는 동일한 추적 ID를 사용하고 웹 앱 클라이언트의 세그먼트를 해당 세그먼트의 상위로 만듭니다.

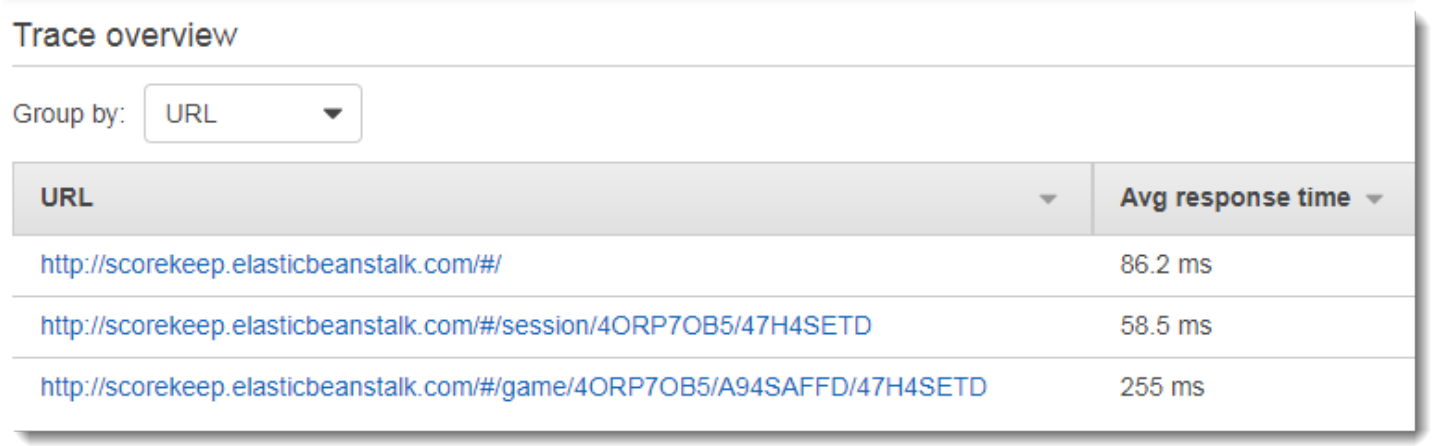
Example **`public/app/services.js`** – Angular 리소스 호출에 대한 세그먼트 레코딩 및 추적 헤더 쓰기

```
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);
      },
    },
...
```

생성된 트레이스 맵에는 웹 앱 클라이언트용 노드가 포함됩니다.



웹 앱의 세그먼트를 포함하는 트레이스는 사용자가 브라우저에서 보는 URL을 표시합니다(/#/로 시작하는 경로). 사용자는 클라이언트 구성 없이 웹 앱이 호출하는 API 리소스의 URL을 가져오기만 합니다(/api/로 시작하는 경로).



작업자 스레드에서 구성된 클라이언트 사용

Scorekeep은 작업자 스레드를 사용하여 사용자가 게임에서 이길 때 Amazon SNS에 알림을 게시합니다. 알림을 게시하는 작업은 결합된 요청 작업의 나머지를 수행할 때보다 더 많은 시간이 걸리며 클라이언트 또는 사용자에게 영향을 주지 않습니다. 따라서 이 작업을 비동기식으로 수행하는 것은 응답 시간을 개선하는 좋은 방법입니다.

그러나 Java용 X-Ray SDK는 스레드가 생성될 때 어떤 세그먼트가 활성화되었는지 알지 못합니다. 따라서 스레드 내에서 계측된 AWS SDK for Java 클라이언트를 사용하려고 하면가 발생하고 스레드가 `SegmentNotFoundException`충돌합니다.

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
...
```

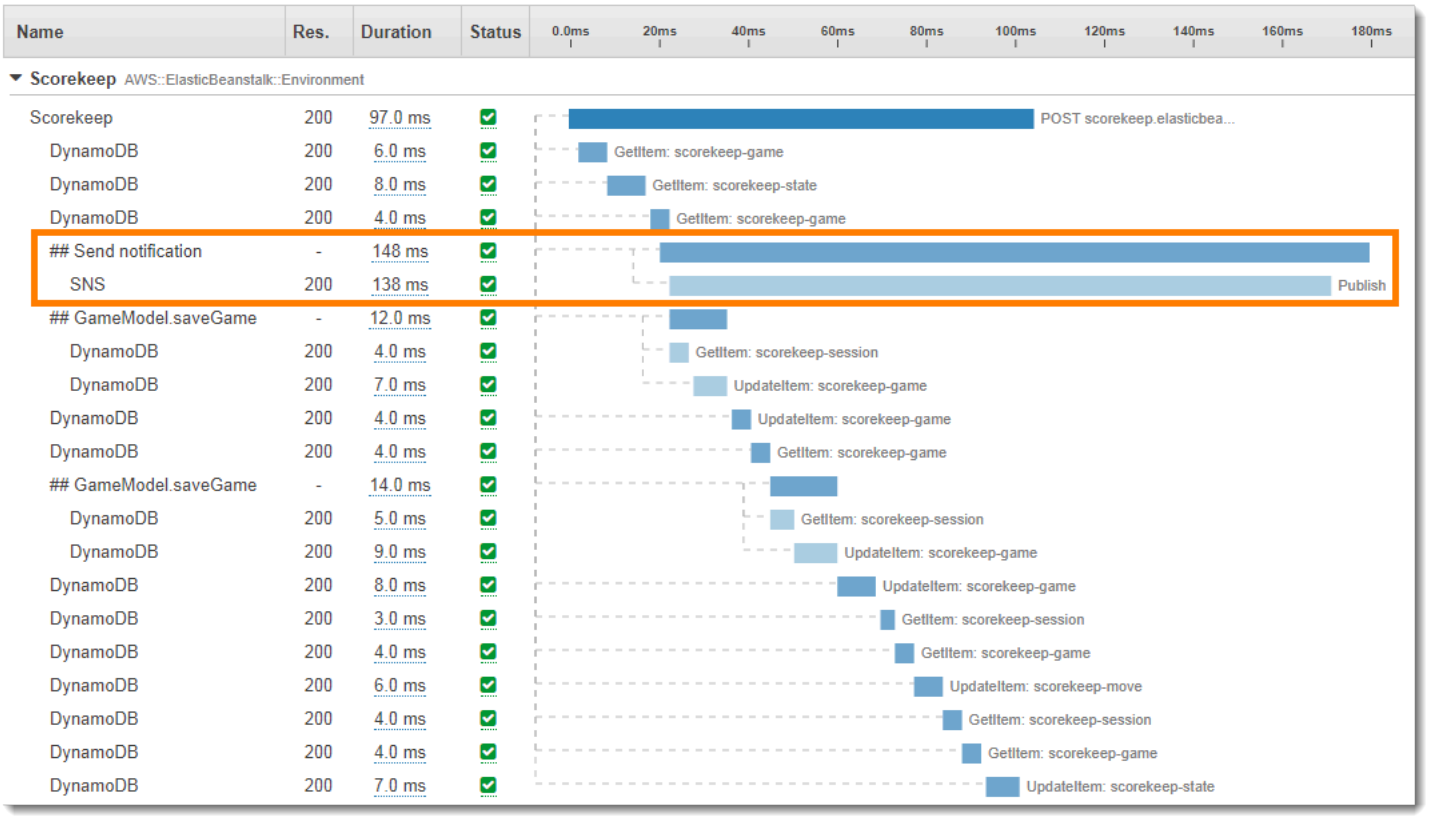
이 문제를 해결하기 위해 애플리케이션에서 `GetTraceEntity`를 사용하여 기본 스레드에서 세그먼트에 대한 참조를 가져오고 `Entity.run()`를 사용하여 세그먼트의 컨텍스트에 대한 액세스 권한으로 작업자 스레드 코드를 안전하게 실행합니다.

Example **src/main/java/scorekeep/MoveFactory.java**— 작업자 스레드에 추적 컨텍스트 전달하기

```
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();
        }
    }
```

이제 Amazon SNS를 직접 호출하기 전에 요청이 해결되기 때문에 애플리케이션은 스레드에 대한 별도의 하위 세그먼트를 생성합니다. 이렇게 하면 X-Ray SDK가 Amazon SNS의 응답을 기록하기 전에 세그먼트를 닫지 않습니다. Scorekeep이 요청을 해결할 때 열려 있는 세그먼트가 없으면 Amazon SNS의 응답이 손실될 수 있습니다.



멀티스레딩에 대한 자세한 내용은 멀티스레드 애플리케이션에서 스레드 간 세그먼트 컨텍스트 전달 단원을 참조하십시오.

AWS X-Ray 데몬

> **Note**
>
> CloudWatch 에이전트를 사용하여 Amazon EC2 인스턴스 및 온프레미스 서버에서 지표, 로그, 추적을 수집할 수 있습니다. CloudWatch 에이전트 버전 1.300025.0 이상은 OpenTelemetry 또는 X-Ray 클라이언트 SDK에서 추적을 수집하여 X-Ray에 전송할 수 있습니다. AWS Distro for OpenTelemetry(ADOT) Collector 또는 X-Ray 데몬 대신 CloudWatch 에이전트를 사용하여 추적을 수집하면 관리하는 에이전트 수를 줄이는 데 도움이 될 수 있습니다. 자세한 내용은 CloudWatch 사용 설명서의 CloudWatch 에이전트 항목을 참조하십시오.

데몬은 UDP 포트 AWS X-Ray 2000에서 트래픽을 수신 대기하고 원시 세그먼트 데이터를 수집하여 AWS X-Ray API에 릴레이하는 소프트웨어 애플리케이션입니다. 데몬은 AWS X-Ray SDKs와 함께 작동하며 SDKs에서 전송한 데이터가 X-Ray 서비스에 도달할 수 있도록 실행 중이어야 합니다. X-Ray 대몬(daemon)은 오픈 소스 프로젝트입니다. 프로젝트를 따르고 GitHub(github.com/aws/aws-xray-daemon)에서 문제 및 풀 요청을 제출할 수 있습니다.

AWS Lambda 및에서 해당 서비스의 X-Ray와의 통합을 AWS Elastic Beanstalk사용하여 데몬을 실행합니다. Lambda는 샘플링된 요청에 대해 함수가 호출될 때마다 자동으로 대몬(daemon)을 실행합니다. Elastic Beanstalk에서는 `XRayEnabled` 구성 옵션을 사용하여 환경 내 인스턴스에서 대몬(daemon)을 실행합니다. 자세한 내용은 단원을 참조하세요.

X-Ray 데몬을 로컬, 온프레미스 또는 다른에서 실행하려면 AWS 서비스다운로드하여 실행한 다음 세그먼트 문서를 X-Ray에 업로드할 수 있는 권한을 부여합니다.

데몬 다운로드

Amazon S3, Amazon ECR 또는 Docker Hub에서 대몬(daemon)을 다운로드한 다음 로컬에서 실행하거나 시작 시 Amazon EC2 인스턴스에 설치할 수 있습니다.

Amazon S3

X-Ray 대몬(daemon) 설치 프로그램 및 실행 파일

- Linux (실행 파일) – `aws-xray-daemon-linux-3.x.zip` (sig)
- Linux (RPM 설치 관리자) – `aws-xray-daemon-3.x.rpm`
- Linux (DEB 설치 관리자) – `aws-xray-daemon-3.x.deb`

- Linux (ARM64, 실행 파일) – aws-xray-daemon-linux-arm64-3.x.zip (sig)
- Linux (ARM64, RPM 설치 관리자) – aws-xray-daemon-arm64-3.x.rpm
- Linux (ARM64, DEB 설치 관리자) – aws-xray-daemon-arm64-3.x.deb
- OS X (실행 파일) – aws-xray-daemon-macos-3.x.zip (sig)
- Windows (실행 파일) – aws-xray-daemon-windows-process-3.x.zip (sig)
- Windows (서비스) – aws-xray-daemon-windows-service-3.x.zip (sig)

이들 링크는 항상 대몬(daemon)의 최신 3.x 릴리스를 가리킵니다. 특정 릴리스를 다운로드하려면 다음을 수행합니다.

- 버전 `3.3.0` 이전 릴리스를 다운로드하려면 `3.x`를 버전 번호로 바꿉니다. 예: `2.1.0`. 버전 `3.3.0` 이전에 사용 가능한 유일한 아키텍처는 `arm64`입니다. 예: `2.1.0` 및 `arm64`.
- `3.3.0` 버전 이후 릴리스를 다운로드하려면 `3.x`는 버전 번호로, `arch`는 아키텍처 유형으로 바꿉니다.

X-Ray 자산은 지원되는 모든 리전에서 버킷에 복제됩니다. 사용자 또는 AWS 리소스와 가장 가까운 버킷을 사용하려면 위 링크의 리전을 해당 리전으로 바꿉니다.

```
https://s3.us-west-2.amazonaws.com/aws-xray-assets.us-west-2/xray-daemon/aws-xray-daemon-3.x.rpm
```

Amazon ECR

버전 3.2.0부터 Amazon ECR에서 대몬(daemon)을 찾을 수 있습니다. 이미지를 가져오기 전에 Amazon ECR 퍼블릭 레지스트리에 docker 클라이언트를 인증해야 합니다.

다음 명령을 실행하여 최신 릴리스된 3.x 버전 태그를 가져옵니다.

```
docker pull public.ecr.aws/xray/aws-xray-daemon:3.x
```

이전 릴리스 또는 알파 릴리스는 `3.x`을 `alpha` 또는 특정 버전 번호로 바꾸어 다운로드할 수 있습니다. 프로덕션 환경에서는 알파 태그가 있는 대몬(daemon) 이미지를 사용하지 않는 것이 좋습니다.

Docker Hub

대몬(daemon)은 Docker Hub에서 찾을 수 있습니다. 최신 릴리스 3.x 버전을 다운로드하려면 다음 명령을 실행하세요.

```
docker pull amazon/aws-xray-daemon:3.x
```

이전 버전의 대몬(daemon)은 `3.x`을 원하는 버전으로 교체하여 릴리스할 수 있습니다.

데몬 아카이브의 서명 확인

ZIP 아카이브로 압축된 데몬 자산에 대해 GPG 서명 파일이 포함됩니다. 퍼블릭 키 위치는 다음과 같습니다. `aws-xray.gpg`.

퍼블릭 키를 사용하여 데몬의 ZIP 아카이브가 원본이며 수정되지 않았는지 확인할 수 있습니다. 먼저 GnuPG를 사용하여 퍼블릭 키를 가져옵니다.

퍼블릭 키 가져오기

1. 퍼블릭 키를 다운로드합니다.

```
$ BUCKETURL=https://s3.us-east-2.amazonaws.com/aws-xray-assets.us-east-2
$ wget $BUCKETURL/xray-daemon/aws-xray.gpg
```

2. 퍼블릭 키를 인증 키로 가져옵니다.

```
$ 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
```

가져온 키를 사용하여 데몬 ZIP 아카이브의 서명을 확인합니다.

아카이브 서명 확인

1. 아카이브 및 서명 파일을 다운로드합니다.

```
$ 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. `gpg --verify`를 실행하여 서명을 확인합니다.

```
$ 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
```

신뢰에 대한 경고를 참조하세요. 사용자 또는 사용자가 신뢰하는 사람이 서명한 키만 신뢰됩니다. 이는 서명이 잘못되었음을 의미하지 않습니다. 단지 해당 사용자가 퍼블릭 키를 확인하지 않은 것입니다.

데몬 실행

명령줄을 사용하여 데몬을 로컬로 실행합니다. -o 옵션을 사용하여 로컬 모드에서 실행하고 -n으로 리전을 설정합니다.

```
~/Downloads$ ./xray -o -n us-east-2
```

플랫폼별 세부 지침은 다음 항목을 참조하십시오.

- Linux (로컬) – Linux에서 X-Ray 대몬(daemon) 실행
- Windows (로컬) – Windows에서 X-Ray 대몬(daemon) 실행
- Elastic Beanstalk – 에서 X-Ray 데몬 실행 AWS Elastic Beanstalk
- Amazon EC2 – Amazon EC2에서 X-Ray 데몬 실행하기
- Amazon ECS – Amazon ECS에서 X-Ray 대몬(daemon) 실행하기

명령줄 옵션 또는 구성 파일을 사용하여 데몬의 동작을 추가로 사용자 지정할 수 있습니다. 세부 정보는 AWS X-Ray 데몬 구성 섹션을 참조하세요.

대몬(daemon)에 X-Ray로 데이터를 전송할 권한 부여

X-Ray 데몬은 AWS SDK를 사용하여 트레이스 데이터를 X-Ray에 업로드하며, 이를 수행하려면 권한이 있는 자격 증명이 필요합니다 AWS .

Amazon EC2에서는 대몬(daemon)이 자동으로 인스턴스의 인스턴스 프로파일 역할을 사용합니다. 대몬을 로컬에서 실행하는 데 필요한 자격 증명에 대한 자세한 내용은 로컬에서 애플리케이션 실행을 참조하세요.

자격 증명 파일, 인스턴스 프로파일 또는 환경 변수 등 둘 이상의 위치에서 자격 증명을 지정한 경우 SDK 공급자 체인이 사용할 자격 증명을 결정합니다. SDK에 자격 증명을 제공하는 방법에 대한 자세한 내용은 AWS SDK for Go 개발자 안내서의 자격 증명 지정 단원을 참조하십시오.

데몬의 자격 증명이 속하는 IAM 역할 또는 사용자는 서비스에 데이터를 기록할 권한이 있어야 합니다.

- Amazon EC2에서 대몬(daemon)을 사용하려면 새 인스턴스 프로파일 역할을 생성하거나 기존 정책에 관리형 정책을 추가합니다.
- Elastic Beanstalk에서 대몬(daemon)을 사용하려면 Elastic Beanstalk 기본 인스턴스 프로파일 역할에 관리형 정책을 추가합니다.
- 대몬(daemon)을 로컬에서 실행하려면 로컬에서 애플리케이션 실행을 참조하십시오.

자세한 내용은 에 대한 자격 증명 및 액세스 관리 AWS X-Ray 단원을 참조하십시오.

X-Ray 대몬(daemon) 로그

데몬은 현재 구성 및 데몬이 보내는 세그먼트에 대한 정보를 출력합니다 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)
```

기본적으로 데몬은 STDOUT에 로그를 출력합니다. 백그라운드에서 데몬을 실행하는 경우 `--log-file` 명령줄 옵션 또는 구성 파일을 사용하여 로그 파일 경로를 설정합니다. 이와 함께 로그 수준을 설정하고 로그 순환을 비활성화할 수 있습니다. 자세한 내용은 AWS X-Ray 데몬 구성 섹션을 참조하세요.

Elastic Beanstalk에서는 플랫폼에 따라 대몬(daemon) 로그의 위치가 설정됩니다. 세부 정보는 에서 X-Ray 데몬 실행 AWS Elastic Beanstalk 섹션을 참조하세요.

AWS X-Ray 데몬 구성

명령줄 옵션 또는 구성 파일을 사용하여 X-Ray 대몬(daemon)의 동작을 사용자 지정할 수 있습니다. 두 가지 방법 모두 대부분의 옵션을 사용할 수 있지만, 구성 파일에만 사용하는 옵션도 있고 명령줄에만 사용하는 옵션도 있습니다.

대몬(daemon)이 X-Ray에 트레이스 데이터를 전송할 때 사용할 리전을 설정하는 `-n` 또는 `--region` 옵션만 알면 시작할 수 있습니다.

```
~/xray-daemon$ ./xray -n us-east-2
```

대몬(daemon)을 Amazon EC2가 아니라 로컬로 실행 중인 경우, `-o` 옵션을 추가해 인스턴스 프로파일 자격 증명 확인을 건너뛰도록 하여 대몬(daemon)을 훨씬 빠르게 준비할 수 있습니다.

```
~/xray-daemon$ ./xray -o -n us-east-2
```

나머지 명령줄 옵션은 로깅을 구성하거나, 다른 포트에서 수신하거나, 데몬이 사용할 수 있는 메모리의 양을 제한하거나, 트레이스 데이터를 다른 계정으로 보내는 역할을 맡는 데 사용할 수 있습니다.

대몬(daemon)에 구성 파일을 전달해 고급 구성 옵션에 액세스하고, X-Ray에 대한 동시 호출 수 제한, 로그 순환 비활성화 및 프록시에 트래픽 전송 등의 작업을 수행할 수 있습니다.

Sections

지원되는 환경 변수

X-Ray 대몬(daemon)은 다음과 같은 환경 변수를 지원합니다.

- `AWS_REGION` – X-Ray 서비스 엔드포인트의 AWS 리전을 지정합니다.
- `HTTPS_PROXY` – 세그먼트 업로드에 이용할 대몬(daemon)의 프록시 주소를 지정합니다. 이것은 DNS 데몬 이름일 수도 있고, 프록시 서버에서 사용하는 IP 주소와 포트 번호일 수도 있습니다.

명령줄 옵션 사용

로컬에서 또는 사용자 데이터 스크립트를 사용하여 데몬을 실행할 때 이러한 옵션을 데몬에 전달합니다.

명령줄 옵션

- `-b`, `--bind` – 다른 UDP 포트에서 세그먼트 문서를 수신 대기합니다.

```
--bind "127.0.0.1:3000"
```

기본값 – 2000.

- -t, --bind-tcp – 다른 TCP 포트에서 X-Ray 서비스에 대한 호출을 수신 대기합니다.

```
-bind-tcp "127.0.0.1:3000"
```

기본값 – 2000.

- -c, --config – 지정된 경로에서 구성 파일을 로드합니다.

```
--config "/home/ec2-user/xray-daemon.yaml"
```

- -f, --log-file – 지정된 파일 경로로 로그를 출력합니다.

```
--log-file "/var/log/xray-daemon.log"
```

- -l, --log-level – 로그 레벨(최대 상세에서 최소 상세까지) dev, debug, info, warn, error, prod.

```
--log-level warn
```

기본값 - prod

- -m, --buffer-memory – 버퍼가 사용할 수 있는 메모리 용량(단위: MB, 최소 3)을 변경합니다.

```
--buffer-memory 50
```

기본값 – 가용 메모리의 1%.

- -o, --local-mode – EC2 인스턴스 메타데이터는 선택하지 마십시오.
- -r, --role-arn – 지정된 IAM 역할을 맡아 세그먼트를 다른 계정으로 업로드합니다.

```
--role-arn "arn:aws:iam::123456789012:role/xray-cross-account"
```

- -a, --resource-arn- 데몬을 실행하는 리소스의 Amazon AWS 리소스 이름(ARN)입니다.
- -p, --proxy-address- 프록시를 AWS X-Ray 통해에 세그먼트를 업로드합니다. 프록시 서버의 프로토콜을 지정해야 합니다.

```
--proxy-address "http://192.0.2.0:3000"
```

- -n, --region – 세그먼트를 특정 리전의 X-Ray 서비스로 전송합니다.
- -v, AWS X-Ray --version- 데몬 버전을 표시합니다.
- -h, --help – 도움말 화면을 표시합니다.

구성 파일 사용

YAML 형식 파일을 사용하여 데몬을 구성할 수도 있습니다. -c 옵션을 사용하여 구성 파일을 데몬으로 전달합니다.

```
~$ ./xray -c ~/xray-daemon.yaml
```

구성 파일 옵션

- TotalBufferSizeMB – 최대 버퍼 크기(단위: MB, 최소 3). 호스트 메모리의 1%를 사용하려면 0을 선택합니다.
- Concurrency - 세그먼트 문서를 업로드 AWS X-Ray 하기 위한에 대한 최대 동시 호출 수입니다.
- Region - 세그먼트를 특정 리전의 AWS X-Ray 서비스로 전송합니다.
- Socket – 대몬(daemon)의 바인딩을 구성합니다.
  - UDPAddress – 대몬(daemon)이 수신 대기하는 포트를 변경합니다.
  - TCPAddress – 다른 TCP 포트에서 X-Ray 서비스에 대한 호출을 수신 대기합니다.
- Logging – 로깅 동작을 구성합니다.
  - LogRotation – 로그 순환을 비활성화하려면 false로 설정합니다.
  - LogLevel – 로그 수준을 가장 자세한 수준부터 가장 낮은 수준까지 변경합니다(dev, debug, info 또는 prod, warn, error, prod). 기본값은 prod이며, info와 동일합니다.
  - LogPath – 지정된 파일 경로로 로그를 출력합니다.
- LocalMode – EC2 인스턴스 메타데이터 확인을 건너뛰려면 true로 설정합니다.
- ResourceARN - 데몬을 실행하는 리소스의 Amazon AWS 리소스 이름(ARN)입니다.
- RoleARN – 지정된 IAM 역할을 맡아 세그먼트를 다른 계정으로 업로드합니다.
- ProxyAddress - 프록시를 AWS X-Ray 통해에 세그먼트를 업로드합니다.
- Endpoint – 대몬(daemon)이 세그먼트 문서를 전송하는 X-Ray 서비스 엔드포인트를 변경합니다.
- NoVerifySSL – TLS 인증서 확인을 비활성화합니다.

- `Version` – 대몬(daemon) 구성 파일 형식 버전. 파일 형식 버전은 필수 입력 사항입니다.

Example xray-daemon.yaml

구성 파일은 데몬의 수신 포트를 3000으로 변경하고, 인스턴스 메타데이터 확인을 끄고, 세그먼트 업로드에 사용할 역할을 설정하고, 리전 및 로깅 옵션을 변경합니다.

```
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
```

로컬에서 X-Ray 대몬(daemon) 실행하기

Linux, MacOS, Windows 또는 Docker 컨테이너에서 AWS X-Ray 데몬을 로컬로 실행할 수 있습니다. 구성된 애플리케이션을 개발 및 테스트할 때 대몬(daemon)을 실행해 X-Ray로 추적 데이터를 전달합니다. 여기의 지침에 따라 데몬을 다운로드한 다음 추출합니다.

로컬에서 실행 중인 경우 데몬은 AWS SDK 자격 증명 파일(`.aws/credentials`사용자 디렉터리) 또는 환경 변수에서 자격 증명을 읽을 수 있습니다. 자세한 내용은 대몬(daemon)에 X-Ray로 데이터를 전송할 권한 부여 단원을 참조하십시오.

이 데몬은 포트 2000에서 UDP 데이터를 수신 대기합니다. 구성 파일 및 명령줄 옵션을 사용하여 포트 및 기타 옵션을 변경할 수 있습니다. 자세한 내용은 AWS X-Ray 데몬 구성 단원을 참조하십시오.

Linux에서 X-Ray 대몬(daemon) 실행

명령줄에서 데몬 실행 파일을 실행할 수 있습니다. `-o` 옵션을 사용하여 로컬 모드에서 실행하고 `-n`으로 리전을 설정합니다.

```
~/xray-daemon$ ./xray -o -n us-east-2
```

데몬을 배경에서 실행하려면 `&`를 사용합니다.

```
~/xray-daemon$ ./xray -o -n us-east-2 &
```

pkill은 배경에서 실행되는 데몬 프로세스를 종료합니다.

```
~$ pkill xray
```

(Docker 컨테이너에서 X-Ray 대몬(daemon) 실행

도커 컨테이너에서 데몬을 로컬로 실행하기 위해 다음 텍스트를 Dockerfile이라는 파일로 저장합니다. Amazon ECR에서 전체 이미지 예제를 다운로드하십시오. 자세한 내용은 대몬(daemon) 다운로드하기를 참조하세요.

Example 도커파일 — 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
```

docker build를 사용하여 컨테이너 이미지를 빌드합니다.

```
~/xray-daemon$ docker build -t xray-daemon .
```

docker run을 사용하여 컨테이너에서 이미지를 실행합니다.

```
~/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
```

이 명령은 다음 옵션을 사용합니다.

- `--attach STDOUT` – 터미널에서 대몬(daemon) 출력을 봅니다.
- `-v ~/.aws/:/root/.aws/:ro` - AWS SDK 자격 증명을 읽을 수 있도록 컨테이너에 `.aws` 디렉터리에 대한 읽기 전용 액세스 권한을 부여합니다.
- `AWS_REGION=us-east-2` – `AWS_REGION` 환경 변수를 설정하여 대몬(daemon)에 어떤 리전을 사용할지 알립니다.
- `--net=host` – 컨테이너를 `host` 네트워크에 연결합니다. 호스트 네트워크의 컨테이너는 포트를 게시하지 않고 서로 통신할 수 있습니다.
- `-p 2000:2000/udp` – 컴퓨터의 UDP 포트 2000을 컨테이너의 동일한 포트로 매핑합니다. 동일한 네트워크의 컨테이너끼리 통신하는 데는 필요 없지만, 도커에서 실행되지 않는 애플리케이션이나 명령줄에서 데몬으로 세그먼트를 전송할 수 있게 됩니다.
- `--name xray-daemon` – 임의의 이름을 생성하는 대신 컨테이너에 `xray-daemon`라는 이름을 지정합니다.
- `-o` (이미지 이름 뒤) – 컨테이너 내에서 대몬(daemon)을 실행하는 진입점에 `-o` 옵션을 추가합니다. 이 옵션은 대몬(daemon)이 Amazon EC2 인스턴스 메타데이터를 읽지 못하도록 로컬 모드에서 실행하도록 지시합니다.

데몬을 중지하려면 `docker stop`을 사용합니다. `Dockerfile`을 변경하여 새 이미지를 빌드한 경우, 같은 이름으로 다른 컨테이너를 생성하려면 기존 컨테이너를 삭제해야 합니다. 컨테이너는 `docker rm`을 사용하여 삭제합니다.

```
$ docker stop xray-daemon
$ docker rm xray-daemon
```

Windows에서 X-Ray 대몬(daemon) 실행

명령줄에서 데몬 실행 파일을 실행할 수 있습니다. `-o` 옵션을 사용하여 로컬 모드에서 실행하고 `-n`으로 리전을 설정합니다.

```
> .\xray_windows.exe -o -n us-east-2
```

PowerShell 스크립트를 사용하여 데몬용 서비스를 생성하고 실행합니다.

Example PowerShell 스크립트 - 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
```

OS X에서 X-Ray 대몬(daemon) 실행

명령줄에서 데몬 실행 파일을 실행할 수 있습니다. -o 옵션을 사용하여 로컬 모드에서 실행하고 -n으로 리전을 설정합니다.

```
~/xray-daemon$ ./xray_mac -o -n us-east-2
```

데몬을 배경에서 실행하려면 &를 사용합니다.

```
~/xray-daemon$ ./xray_mac -o -n us-east-2 &
```

터미널이 닫힐 때 데몬이 종료되는 것을 방지하려면 nohup을 사용합니다.

```
~/xray-daemon$ nohup ./xray_mac &
```

에서 X-Ray 데몬 실행 AWS Elastic Beanstalk

애플리케이션에서 로 트레이스 데이터를 릴레이하려면 Elastic Beanstalk 환경의 Amazon EC2 인스턴스에서 X-Ray 데몬을 실행할 AWS X-Ray수 있습니다. 지원되는 플랫폼 목록은 AWS Elastic Beanstalk 개발자 안내서의 AWS X-Ray 디버깅 구성을 참조하십시오.

> **Note**
>
> 데몬은 권한을 위해 환경의 인스턴스 프로파일을 사용합니다. Elastic Beanstalk 인스턴스 프로파일에 권한을 추가하는 자세한 내용은 대몬(daemon)에 X-Ray로 데이터를 전송할 권한 부여 섹션을 참조하십시오.

Elastic Beanstalk 플랫폼에는 대몬(daemon)을 자동으로 실행하도록 설정할 수 있는 구성 옵션이 있습니다. 소스 코드의 구성 파일을 사용하거나 Elastic Beanstalk 콘솔에서 옵션을 선택하여 대몬(daemon)을 활성화할 수 있습니다. 구성 옵션을 활성화하면 데몬이 인스턴스에 설치되고 서비스로 실행됩니다.

Elastic Beanstalk 플랫폼에 포함된 버전이 최신 버전이 아닐 수 있습니다. 해당 플랫폼 구성에서 사용 가능한 데몬의 버전을 확인하려면 지원되는 플랫폼 항목을 참조하십시오.

Elastic Beanstalk는 멀티컨테이너 도커(Amazon ECS) 플랫폼에서 X-Ray 대몬(daemon)을 제공하지 않습니다.

Elastic Beanstalk X-Ray 통합을 사용하여 X-Ray 대몬(daemon) 실행하기

콘솔을 사용하여 X-Ray 통합을 켜거나 구성 파일을 사용하여 애플리케이션 소스 코드에서 구성합니다.

Elastic Beanstalk 콘솔에서 X-Ray 대몬(daemon)을 활성화하려면

1. Elastic Beanstalk 콘솔을 엽니다.
2. 사용 중인 환경의 관리 콘솔로 이동합니다.
3. 구성을 선택합니다.
4. [Software Settings]를 선택합니다.
5. [X-Ray daemon]에서 [Enabled]를 선택합니다.
6. 적용을 선택합니다.

구성 파일을 소스 코드에 포함시켜 구성을 환경 사이에서 이동할 수 있습니다.

Example .ebextensions/xray-daemon.config

```
option_settings:
  aws:elasticbeanstalk:xray:
    XRayEnabled: true
```

Elastic Beanstalk은 구성 파일을 대몬(daemon)에 전달하고 로그를 표준 위치로 출력합니다.

Windows Server 플랫폼

- 구성 파일 – `C:\Program Files\Amazon\XRay\cfg.yaml`
- 로그 – `c:\Program Files\Amazon\XRay\logs\xray-service.log`

Linux 플랫폼

- 구성 파일 – `/etc/amazon/xray/cfg.yaml`
- 로그 – `/var/log/xray/xray.log`

Elastic Beanstalk는 AWS Management Console 또는 명령줄에서 인스턴스 로그를 가져오는 도구를 제공합니다. 구성 파일을 포함하는 작업을 추가하여 Elastic Beanstalk이 X-Ray 대몬(daemon) 로그를 포함하도록 지정할 수 있습니다.

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
```

자세한 내용은 AWS Elastic Beanstalk 개발자 안내서의 Elastic Beanstalk 환경에서 Amazon EC2 인스턴스의 로그 보기를 참조하세요.

수동으로 X-Ray 대몬(daemon) 다운로드 및 실행하기 (고급)

플랫폼 구성에 X-Ray 대몬(daemon)을 사용할 수 없는 경우 Amazon S3에서 다운로드하여 구성 파일과 함께 실행할 수 있습니다.

Elastic Beanstalk 구성 파일을 사용하여 대몬(daemon)을 다운로드하고 실행합니다.

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
```

또한 이 예제에서는 대몬(daemon)의 로그 파일을 Elastic Beanstalk 테일 로그 작업에 추가하여 콘솔 또는 Elastic Beanstalk 명령줄 인터페이스(EB CLI)로 로그를 요청할 때 포함되도록 합니다.

Amazon EC2에서 X-Ray 데몬 실행하기

Amazon EC2의 다음 운영 체제에서 X-Ray 대몬(daemon)을 실행할 수 있습니다:

- Amazon Linux
- Ubuntu
- Windows Server(2012 R2 이상)

인스턴스 프로파일을 사용하여 대몬(daemon)이 추적 데이터를 X-Ray에 업로드할 권한을 부여합니다. 자세한 내용은 대몬(daemon)에 X-Ray로 데이터를 전송할 권한 부여 단원을 참조하십시오.

사용자 데이터 스크립트를 사용하여 인스턴스를 시작할 때 자동으로 대몬(daemon)을 실행할 수 있습니다.

Example 사용자 데이터 스크립트 – 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 데이터 스크립트 사용 - 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>
```

Amazon ECS에서 X-Ray 대몬(daemon) 실행하기

Amazon ECS에서 X-Ray 대몬(daemon)을 실행하는 도커 이미지를 생성하여 도커 이미지 리포지토리에 업로드한 다음 Amason ECS 클러스터에 배포합니다. 작업 정의 파일의 포트 매핑 및 네트워크 모드 설정을 사용하여 애플리케이션이 데몬 컨테이너와 통신하도록 할 수 있습니다.

공식 도커 이미지 사용

X-Ray는 애플리케이션과 함께 배포할 수 있는 Amazon ECR의 도커 컨테이너 이미지를 제공합니다. 자세한 내용은 대몬(daemon) 다운로드하기를 참조하세요.

Example 태스크 정의

```
{
  "name": "xray-daemon",
  "image": "amazon/aws-xray-daemon",
  "cpu": 32,
  "memoryReservation": 256,
  "portMappings" : [
      {
          "hostPort": 0,
          "containerPort": 2000,
          "protocol": "udp"
```

```
        }
      ]
    }
```

도커 이미지 생성 및 빌드

사용자 지정 구성의 경우 자체 도커 이미지를 정의해야 할 수 있습니다.

작업 역할에 관리형 정책을 추가해 대몬(daemon)에 트레이스 데이터를 X-Ray에 업로드할 권한을 부여합니다. 자세한 내용은 대몬(daemon)에 X-Ray로 데이터를 전송할 권한 부여 단원을 참조하십시오.

다음 Dockerfile 중 하나를 사용하여 데몬을 실행하는 이미지를 생성합니다.

Example 도커파일 — 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**
>
> 다중 컨테이너 환경의 루프백을 수신 대기하는 바인딩 주소를 지정하려면 플래그 `-t` 및 `-b`가 필요합니다.

Example Dockerfile — Ubuntu

데비안 파생 버전의 경우, 설치 프로그램을 다운로드할 때 문제가 발생하지 않도록 CA(인증 기관) 인증서도 설치해야 합니다.

```
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
```

사용하는 네트워크 모드에 따라 작업 정의의 구성이 달라집니다. 기본값은 브리지 네트워킹이며, 기본 VPC에서 사용할 수 있습니다. 브리지 네트워크에서 AWS_XRAY_DAEMON_ADDRESS 환경 변수를 설정하여 SDK에 참조할 컨테이너 포트를 알리고 호스트 포트를 설정합니다. 예를 들어, UDP 포트 2000을 게시하고 애플리케이션 컨테이너에서 데몬 컨테이너로 이어지는 링크를 생성합니다.

Example 태스크 정의

```
    {
      "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"
      ]
    }
```

VPC의 프라이빗 서브넷에서 클러스터를 실행하는 경우, awsvpc 네트워크 모드를 사용하여 컨테이너에 ENI(탄력적 네트워크 인터페이스)를 연결할 수 있습니다. 이로써 링크 사용을 피할 수 있습니다. 포트 매핑, 링크 및 AWS_XRAY_DAEMON_ADDRESS 환경 변수에서 호스트 포트를 생략합니다.

Example VPC 작업 정의

```
{
    "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
                }
            ]
        }
```

```
    ]
}
```

Amazon ECS 콘솔에서 명령줄 옵션 구성

명령줄 옵션은 이미지의 구성 파일에서 충돌하는 모든 값을 재정의합니다. 명령줄 옵션은 일반적으로 로컬 테스트에 사용되지만 환경 변수를 설정하는 동안 편의를 위해 사용하거나 시작 프로세스를 제어하기 위해 사용할 수도 있습니다.

명령줄 옵션을 추가하면 컨테이너에 전달되는 Docker `CMD`가 업데이트됩니다. 자세한 내용은 Docker 실행 참조를 확인하십시오.

명령줄 옵션 설정

1. https://console.aws.amazon.com/ecs/에서 Amazon ECS 클래식 콘솔을 엽니다.
2. 탐색 모음에서 태스크 정의가 들어 있는 리전을 선택합니다.
3. 탐색 창에서 태스크 정의를 선택합니다.
4. 작업 정의 페이지에서 개정할 작업 정의 왼쪽의 확인란을 선택한 후 새 개정 생성을 선택합니다.
5. 작업 정의의 새 개정 생성 페이지에서 컨테이너를 선택합니다.
6. 환경 섹션에서 쉼표로 구분된 명령줄 옵션 목록을 명령 필드에 추가합니다.
7. 업데이트를 선택합니다.
8. 정보를 확인한 후 생성(Create)을 선택합니다.

다음 예제에서는 `RoleARN` 옵션에 대해 쉼표로 구분된 명령줄 옵션을 작성하는 방법을 보여줍니다. `RoleARN` 옵션은 지정된 IAM 역할을 맡아 세그먼트를 다른 계정으로 업로드합니다.

Example

```
--role-arn, arn:aws:iam::123456789012:role/xray-cross-account
```

X-Ray에서 사용 가능한 명령줄 옵션에 대한 자세한 내용은 AWS X-Ray 데몬 구성을 참조하세요.

다른 AWS X-Ray 와 통합 AWS 서비스

수신 요청에 헤더를 샘플링하고 추가하고, X-Ray 데몬을 실행하고, 추적 데이터를 X-Ray로 자동 전송하는 등 다양한 수준의 X-Ray 통합을 AWS 서비스 제공합니다. X-Ray와의 통합에는 다음이 포함될 수 있습니다.

- 활성 계측 – 수신 요청을 샘플링하고 구성합니다.
- 수동 계측 – 다른 서비스에 의해 샘플링된 요청을 구성합니다.
- 요청 추적 – 모든 수신 요청에 추적 헤더를 추가하고 다운스트림으로 전파합니다.
- 도구 – X-Ray 대몬(daemon)을 실행하여 X-Ray SDK로부터 세그먼트를 수신합니다.

> **Note**
>
> X-Ray SDKs 포함되어 있습니다 AWS 서비스. 예를 들어 Java Elastic Beanstalk용 X-Ray SDK 플러그인을 사용하여 환경 이름과 ID 등 애플리케이션이 실행되는 Elastic Beanstalk 환경에 대한 정보를 추가할 수 있습니다.

다음은 X-Ray와 통합된의 몇 가지 예 AWS 서비스 입니다.

- AWS Distro for OpenTelemetry(ADOT) - ADOT를 사용하면 엔지니어가 애플리케이션을 한 번 계측하고 상관관계가 있는 지표와 추적을 Amazon CloudWatch, AWS X-Ray, Amazon OpenSearch Service, Amazon Managed Service for Prometheus를 포함한 여러 AWS 모니터링 솔루션으로 전송할 수 있습니다.
- AWS Lambda - 모든 런타임에서 수신되는 요청을 능동적 및 수동적으로 계측합니다.는 트레이스 맵에 두 개의 노드를 AWS Lambda 추가합니다. 하나는 AWS Lambda 서비스용이고 다른 하나는 함수용입니다. 계측을 활성화하면는 X-Ray SDK와 함께 사용할 Java 및 Node.js 런타임에서 X-Ray 데몬 AWS Lambda 도 실행합니다.
- Amazon API Gateway — 액티브 및 패시브 계측. API 게이트웨이는 샘플링 규칙을 사용하여 기록할 요청을 결정하고 서비스 맵에 게이트웨이 단계에 대한 노드를 추가합니다.
- AWS Elastic Beanstalk – 도구. Elastic Beanstalk는 다음 플랫폼에 대한 X-Ray 대몬(daemon)을 포함합니다.
  - Java SE – 2.3.0 이상 구성
  - Tomcat – 2.4.0 이상 구성

  - Node.js – 3.2.0 이상 구성
  - Windows Server – 2016년 12월 9일 이후 릴리스된 Windows Server Core 이외의 모든 구성.

  Elastic Beanstalk 콘솔을 사용하여 이러한 플랫폼에서 대몬(daemon)을 실행하도록 Elastic Beanstalk에 지시하거나 `aws:elasticbeanstalk:xray` 네임스페이스에서 `XRayEnabled` 옵션을 사용할 수 있습니다.

- Elastic Load Balancing – Application Load Balancer에서 추적을 요청합니다. Application Load Balancer는 트레이스 ID를 요청 헤더에 전달한 다음 요청 헤더를 대상 그룹으로 전송합니다.
- Amazon EventBridge — 수동 계측. EventBridge에 이벤트를 게시하는 서비스가 X-Ray SDK로 계측되는 경우, 이벤트 타겟은 추적 헤더를 수신하고 원래 추적 ID를 계속 전파할 수 있습니다.
- Amazon Simple Notification Service – 수동 계측. Amazon SNS 게시자가 X-Ray SDK로 클라이언트를 추적하는 경우, 구독자는 추적 헤더를 검색하여 동일한 추적 ID를 가진 게시자로부터 원본 추적을 계속 전파할 수 있습니다.
- Amazon Simple Queue Service – 수동 계측. X-Ray SDK를 사용하여 요청을 추적하는 서비스가 있는 경우, Amazon SQS는 추적 헤더를 전송하고 발신자가 소비자에게 보낸 원본 추적에 일정한 추적 ID를 계속해서 전파합니다.
- Amazon Bedrock AgentCore - AgentCore는 X-Ray 통합을 통한 분산 추적을 지원하므로 에이전트 애플리케이션을 통과하는 요청을 추적할 수 있습니다. AgentCore 리소스에 대한 관찰성을 활성화하면 서비스 경계에 트레이스 컨텍스트를 전파하고 AI 에이전트 및 도구의 성능에 대한 가시성을 확보할 수 있습니다.

다음 주제 중에서 선택하여 전체 통합 세트를 살펴보세요 AWS 서비스.

주제

- Amazon Bedrock AgentCore 및 AWS X-Ray
- Amazon Elastic Compute Cloud 및 AWS X-Ray
- Amazon SNS 및 AWS X-Ray
- Amazon SQS 및 AWS X-Ray
- Amazon S3 및 AWS X-Ray
- AWS Distro for OpenTelemetry 및 AWS X-Ray
- 를 사용하여 X-Ray 암호화 구성 변경 사항 추적 AWS Config
- AWS AppSync 그리고 AWS X-Ray
- 에 대한 Amazon API Gateway 활성 추적 지원 AWS X-Ray

- Amazon EC2 및 AWS App Mesh
- AWS App Runner 및 X-Ray
- 를 사용하여 X-Ray API 호출 로깅 AWS CloudTrail
- CloudWatch와 X-Ray 통합
- AWS Elastic Beanstalk 그리고 AWS X-Ray
- Elastic Load Balancing 및 AWS X-Ray
- Amazon EventBridge 및 AWS X-Ray
- AWS Lambda 그리고 AWS X-Ray
- AWS Step Functions 그리고 AWS X-Ray

Amazon Bedrock AgentCore 및 AWS X-Ray

Amazon Bedrock AgentCore는와 통합되어 AI 에이전트 및 도구를 위한 분산 추적 기능을 AWS X-Ray 제공합니다. 이 통합을 통해 에이전트 애플리케이션을 통해 요청이 흐를 때 요청을 추적하여 성능 병목 현상을 식별하고 문제를 해결할 수 있습니다.

AgentCore는 X-Ray 통합을 통한 분산 추적을 지원하므로 AI 에이전트 및 도구의 성능을 모니터링할 수 있습니다. AgentCore 리소스에 대한 관찰성을 활성화하면 서비스 경계에 트레이스 컨텍스트를 전파하고 에이전트가 다른 AWS 서비스와 상호 작용하는 방식을 파악할 수 있습니다. 자세한 내용은 Amazon Bedrock AgentCore를 참조하세요.

AgentCore는 다음과 같은 X-Ray 기능을 지원합니다.

- 추적 컨텍스트를 다운스트림 서비스로 전파
- AWS Distro for OpenTelemetry(ADOT) SDK를 사용한 사용자 지정 계측

AgentCore를 사용하여 X-Ray 설정

AgentCore에서 X-Ray를 사용하려면 AWS 계정에서 CloudWatch 트랜잭션 검색을 활성화해야 합니다. 이 설정은 AgentCore가 트레이스 데이터를 X-Ray로 전송할 수 있도록 하는 일회성 설정입니다. 자세한 내용은 트랜잭션 검색 활성화를 참조하세요.

AgentCore의 관찰성 설정에 대한 자세한 내용은 Amazon Bedrock AgentCore 에이전트 또는 도구에 관찰성 추가를 참조하세요.

AgentCore에서 트레이스 헤더 사용

AgentCore는 분산 추적을 위한 X-Ray 추적 헤더 형식을 지원합니다. AgentCore에 대한 요청에 `X-Amzn-Trace-Id` 헤더를 포함하여 서비스 경계 전반에 걸쳐 추적 컨텍스트를 유지할 수 있습니다.

Amazon Elastic Compute Cloud 및 AWS X-Ray

사용자 데이터 스크립트를 사용하여 Amazon EC2 인스턴스에서 X-Ray 대몬(daemon)을 설치하고 실행할 수 있습니다. 자세한 내용은 Amazon EC2에서 X-Ray 데몬 실행하기 섹션을 참조하세요.

인스턴스 프로파일을 사용하여 대몬(daemon)이 추적 데이터를 X-Ray에 업로드할 권한을 부여합니다. 자세한 내용은 대몬(daemon)에 X-Ray로 데이터를 전송할 권한 부여 단원을 참조하십시오.

Amazon SNS 및 AWS X-Ray

Amazon Simple Notification Service(SNS)와 AWS X-Ray 함께를 사용하여 SNS 주제를 통해 SNS 지원 구독 서비스로 이동하는 요청을 추적하고 분석할 수 있습니다.Amazon SNS Amazon SNS와 함께 X-Ray 추적을 사용하여 토픽에서 요청이 소요되는 시간, 각 토픽의 구독에 메시지를 전달하는 데 걸리는 시간 등 메시지와 해당 백엔드 서비스의 지연 시간을 분석할 수 있습니다. Amazon SNS는 표준 및 FIFO 주제에 대해 X-Ray 추적을 지원합니다.

이미 X-Ray로 계측된 서비스에서 Amazon SNS 주제에 게시하는 경우, Amazon SNS는 게시자에서 구독자에게 추적 컨텍스트를 전달합니다. 또한 활성 추적을 활성화하여 계측된 SNS 클라이언트에서 게시된 메시지에 대해 Amazon SNS 구독에 대한 세그먼트 데이터를 X-Ray로 전송할 수 있습니다. Amazon SNS 콘솔을 사용하거나 Amazon SNS API 또는 CLI를 사용하여 Amazon SNS 주제에 대한 활성 추적을 활성화합니다. SNS 클라이언트 계측에 대한 자세한 내용은 애플리케이션 계측을 참조하십시오.

Amazon SNS 활성 추적 구성하기

Amazon SNS 콘솔 또는 AWS CLI 또는 SDK를 사용하여 Amazon SNS 활성 추적을 구성할 수 있습니다.

Amazon SNS 콘솔을 사용할 때 Amazon SNS는 SNS가 X-Ray를 직접 호출하는 데 필요한 권한을 생성하려고 시도합니다. X-Ray 리소스 정책을 수정할 수 있는 충분한 권한이 없는 경우 시도가 거부될 수 있습니다. 이러한 권한에 대한 자세한 내용은 Amazon Simple Notification Service 개발자 가이드에서 Amazon SNS 신원 및 액세스 관리 및 Amazon SNS 제어 예시를 참조하세요. Amazon SNS 콘솔을 사용하여 액티브 추적을 켜는 방법에 대한 자세한 내용은 Amazon 단순 알림 서비스 개발자 가이드의 Amazon SNS에서 액티브 추적 사용 설정 항목을 참조하세요.

AWS CLI 또는 SDK를 사용하여 활성 추적을 켤 때는 리소스 기반 정책을 사용하여 권한을 수동으로 구성해야 합니다. PutResourcePolicy를 사용하여 Amazon SNS가 X-Ray로 추적을 전송할 수 있도록 필요한 리소스 기반 정책으로 X-Ray를 구성합니다.

Example Amazon SNS 활성 추적에 대한 X-Ray 리소스 기반 정책 예시

이 예시 정책 문서는 추적 데이터를 X-Ray로 전송하는 데 필요한 Amazon SNS의 권한을 지정합니다.

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

CLI를 사용하여 추적 데이터를 X-Ray로 전송할 수 있는 Amazon SNS 권한을 부여하는 리소스 기반 정책을 생성합니다.

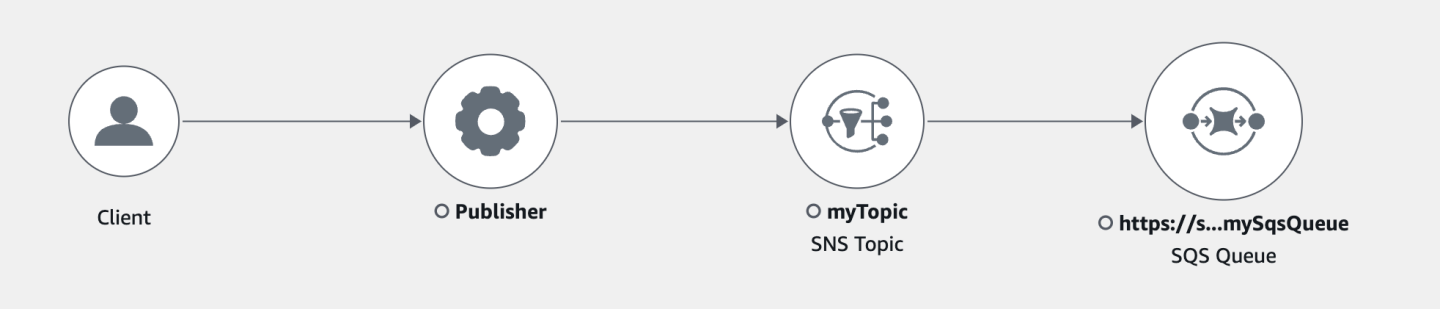
```
aws xray put-resource-policy --policy-name MyResourcePolicy --policy-document
 '{ "Version": "2012-10-17", "Statement": [ { "Sid": "SNSAccess", "Effect": "Allow",
 "Principal": { "Service": "sns.amazonaws.com" }, "Action": [ "xray:PutTraceSegments",
 "xray:GetSamplingRules", "xray:GetSamplingTargets" ], "Resource": "*",
```

```
"Condition": { "StringEquals": { "aws:SourceAccount": "account-id" }, "StringLike":
{ "aws:SourceArn": "arn:partition:sns:region:account-id:topic-name" } } } ] }'
```

이러한 예제를 사용하려면 *partition*, *account-id*, 및 *regiontopic-name*를 특정 AWS 파티션, 리전, 계정 ID 및 Amazon SNS 주제 이름으로 바꿉니다. 모든 Amazon SNS 주제가 추적 데이터를 X-Ray로 전송하도록 권한을 부여하려면 주제 이름을 *로 바꾸십시오.

X-Ray 콘솔에서 Amazon SNS 게시자 및 구독자 추적 보기

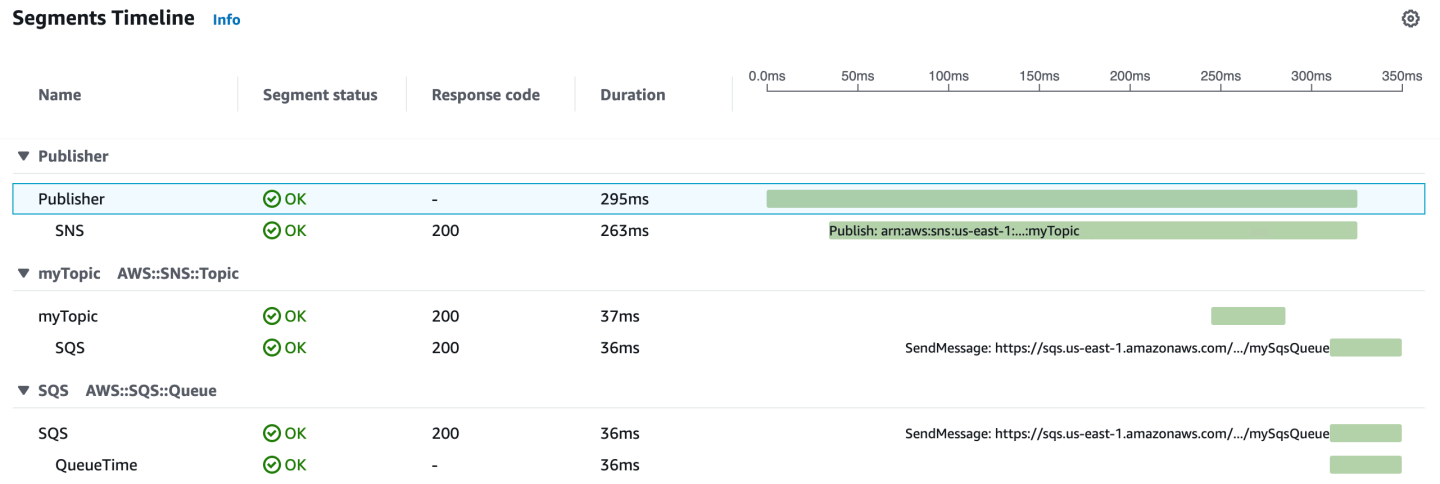
X-Ray 콘솔을 사용하여 Amazon SNS 게시자 및 구독자의 연결된 상태를 표시하는 트레이스 맵과 트레이스 세부 정보를 볼 수 있습니다. 주제에 대해 Amazon SNS 활성 추적이 켜져 있으면 X-Ray 트레이스 맵과 트레이스 세부 정보 맵에 Amazon SNS 게시자, Amazon SNS 주제 및 다운스트림 구독자에 대한 연결된 노드가 표시됩니다.



Amazon SNS 게시자와 구독자를 포함하는 트레이스를 선택하면 X-Ray 트레이스 세부 정보 페이지에 트레이스 세부 정보 맵과 세그먼트 타임라인이 표시됩니다.

Example Amazon SNS 게시자 및 구독자가 포함된 예제 타임라인

이 예는 Amazon SQS 게시자가 Amazon SNS 주제에 메시지를 보내는 타임라인을 보여주며, 이 메시지는 Amazon SQS 구독자가 처리합니다.

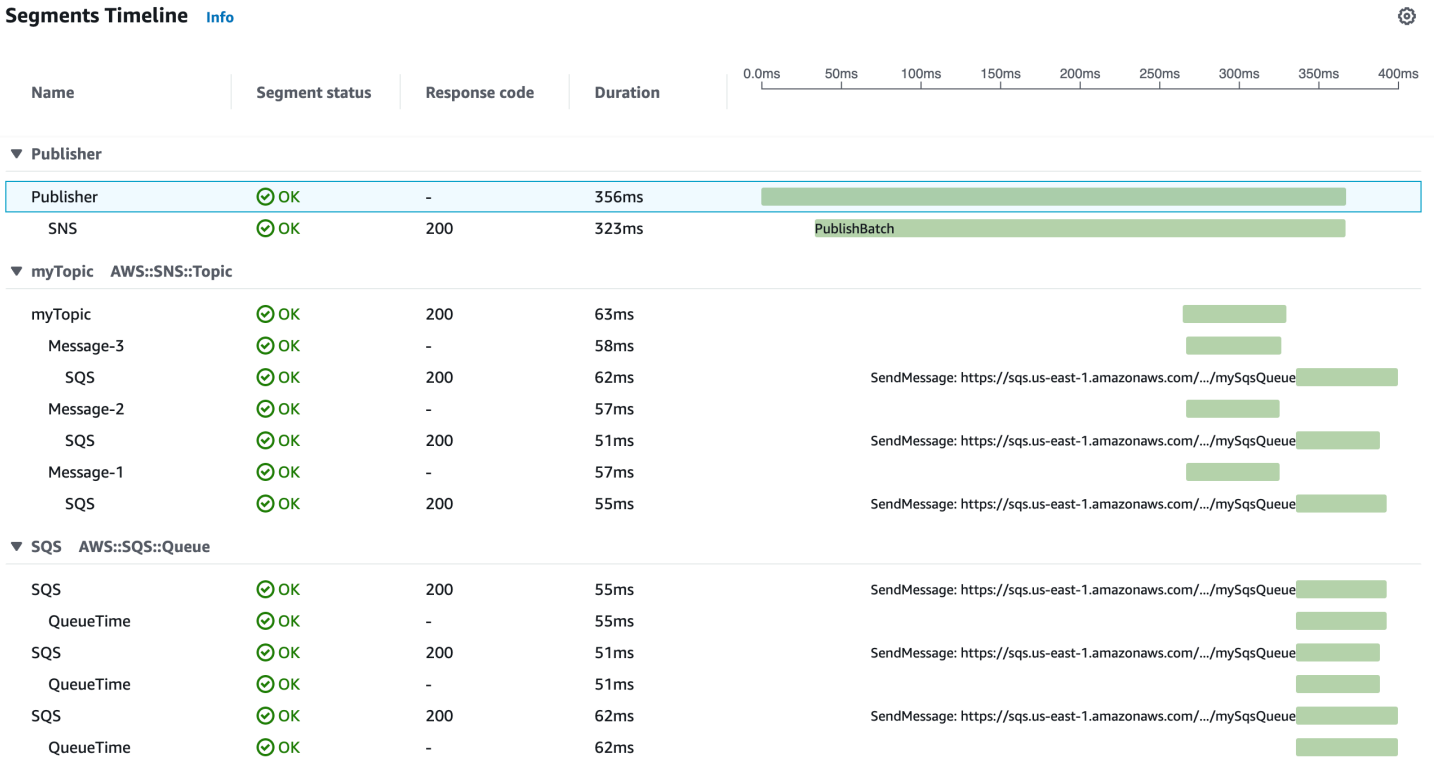


위의 예제 타임라인은 Amazon SNS 메시지 흐름에 대한 세부 정보를 제공합니다:

- SNS 세그먼트는 클라이언트의 `Publish` API 직접 호출 왕복 시간을 나타냅니다.
- myTopic 세그먼트는 게시 요청에 대한 Amazon SNS 응답의 지연 시간을 나타냅니다.
- SQS 하위 세그먼트는 Amazon SNS가 Amazon SQS 대기열에 메시지를 게시하는 데 걸리는 왕복 시간을 나타냅니다.
- myTopic 세그먼트와 SQS 하위 세그먼트 사이의 시간은 Amazon SNS 시스템에서 메시지가 머무는 시간을 나타냅니다.

Example 일괄 처리된 Amazon SNS 메시지가 포함된 타임라인 예시

여러 개의 Amazon SNS 메시지가 하나의 추적 내에 일괄 처리되는 경우, 세그먼트 타임라인에는 처리된 각 메시지를 나타내는 세그먼트가 표시됩니다.

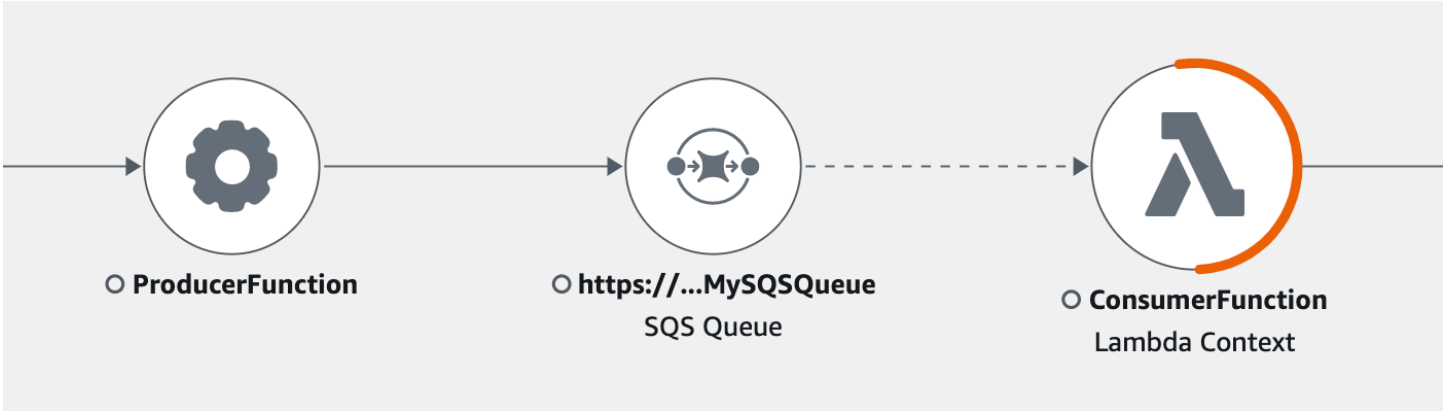


Amazon SQS 및 AWS X-Ray

AWS X-Ray 는 Amazon Simple Queue Service(Amazon SQS)와 통합되어 Amazon SQS 대기열을 통해 전달되는 메시지를 추적합니다. X-Ray SDK를 사용하여 요청을 추적하는 서비스가 있는 경우, Amazon SQS는 추적 헤더를 전송하고 발신자가 소비자에게 보낸 원본 추적에 일정한 추적 ID를 계속

해서 전파합니다. 사용자는 이러한 추적의 연속성 덕분에 다운스트림 서비스 전체에 걸쳐 추적, 분석 및 디버깅할 수 있습니다.

AWS X-Ray 는 Amazon SQS 및를 사용하여 이벤트 기반 애플리케이션 추적을 지원합니다 AWS Lambda. CloudWatch 콘솔을 사용하면 각 요청이 Amazon SQS로 대기열에 추가되고 다운스트림 Lambda 함수에 의해 처리될 때 연결된 보기를 볼 수 있습니다. 업스트림 메시지 생산자의 트레이스가 다운스트림 Lambda 함수의 트레이스에 자동으로 연결되므로 전체 애플리케이션을 종합적으로 파악할 수 있습니다. 자세한 내용은 이벤트 중심 애플리케이션 추적을 참조하세요.



Amazon SQS는 다음과 같은 추적 헤더 계측을 지원합니다.

- 기본 HTTP 헤더 - SDK를 통해 Amazon SQS를 호출하면 X-Ray AWS SDK가 추적 헤더를 HTTP 헤더로 자동으로 채웁니다. 기본 추적 헤더는 `X-Amzn-Trace-Id`로 입력되며, `SendMessage` 또는 `SendMessageBatch` 요청에 포함된 모든 메시지에 해당합니다. 기본 HTTP 헤더에 대한 자세한 내용은 추적 헤더 단원을 참조하십시오.
- **`AWSTraceHeader`** 시스템 속성 - `AWSTraceHeader`는 대기열의 메시지와 함께 X-Ray 추적 헤더를 전달하기 위해 Amazon SQS에서 사용하는 메시지 시스템 속성입니다. 예를 들어 새로운 언어에 대한 추적 SDK를 빌드할 때와 같이 X-Ray SDK를 통한 자동 계측이 제공되지 않는 경우에도 `AWSTraceHeader`를 사용할 수 있습니다. 두 가지 헤더 계측을 설정하고 나면 그 메시지 시스템 속성이 HTTP 추적 헤더를 덮어씁니다.

Amazon EC2에서 실행하는 경우 Amazon SQS는 한 번에 하나의 메시지 처리를 지원합니다. 이는 온프레미스 호스트에서 실행되고 AWS Fargate Amazon ECS 또는와 같은 컨테이너 서비스를 사용할 때 적용됩니다 AWS App Mesh.

추적 헤더는 Amazon SQS 메시지 크기 할당량 및 메시지 속성 할당량에서 제외됩니다. 즉, X-Ray 추적을 활성화해도 Amazon SQS 할당량을 초과하지 않습니다. AWS 할당량에 대한 자세한 내용은 Amazon SQS 할당량을 참조하세요.

HTTP 추적 헤더 전송

Amazon SQS의 발신자 구성 요소는 `SendMessageBatch` 또는 `SendMessage` 직접 호출을 통해 추적 헤더를 자동 전송할 수 있습니다. AWS SDK 클라이언트가 계측되면 X-Ray SDK를 통해 지원되는 모든 언어를 통해 클라이언트를 자동으로 추적할 수 있습니다. 해당 서비스(예: Amazon S3 버킷 또는 Amazon SQS 대기열) 내에서 액세스하는 추적 AWS 서비스 및 리소스는 X-Ray 콘솔의 추적 맵에 다운스트림 노드로 표시됩니다.

원하는 언어로 AWS SDK 호출을 추적하는 방법을 알아보려면 지원되는 SDKs

- Go – Go용 X-Ray AWS SDK를 사용하여 SDK 호출 추적
- Java – Java용 X-Ray AWS SDK를 사용하여 SDK 호출 추적
- Node.js – Node.js용 X-Ray AWS SDK를 사용하여 SDK 호출 추적
- Python – Python용 X-Ray AWS SDK를 사용하여 SDK 호출 추적
- Ruby – Ruby용 X-Ray AWS SDK를 사용하여 SDK 호출 추적
- .NET – .NET용 X-Ray AWS SDK를 사용하여 SDK 호출 추적

추적 헤더 검색 및 추적 컨텍스트 복구

Lambda 다운스트림 소비자를 사용하는 경우 추적 컨텍스트 전파가 자동으로 이루어집니다. 다른 Amazon SQS 소비자와 컨텍스트 전파를 계속하려면 수신자 구성 요소에 대한 핸드오프를 수동으로 계측해야 합니다.

추적 컨텍스트를 복구하는 단계는 크게 세 단계로 나뉩니다.

- `ReceiveMessage` API를 호출하여 `AWSTraceHeader` 속성에 대한 대기열에서 메시지를 수신합니다.
- 속성에서 추적 헤더를 검색합니다.
- 헤더에서 추적 ID를 복구합니다. 원한다면 해당 세그먼트에 지표를 더 추가합니다.

다음은 Java용 X-Ray SDK로 작성된 구현 예제입니다.

Example : 추적 헤더 검색 및 추적 컨텍스트 복구

```
// 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 및 AWS X-Ray

AWS X-Ray 는 Amazon S3와 통합되어 애플리케이션의 S3 버킷을 업데이트하기 위한 업스트림 요청을 추적합니다. 서비스가 X-Ray SDK를 사용하여 요청을 추적하는 경우, Amazon S3는 추적 헤더를 AWS Lambda, Amazon SQS, Amazon SNS와 같은 다운스트림 이벤트 구독자에게 보낼 수 있습니다. X-Ray는 Amazon S3 이벤트 알림에 대한 추적 메시지를 활성화합니다.

X-Ray 트레이스 맵을 사용하여 Amazon S3와 애플리케이션에서 사용하는 다른 서비스의 연결을 확인할 수 있습니다. 또한 콘솔을 사용하여 평균 대기 시간 및 실패율과 같은 메트릭을 볼 수 있습니다. X-Ray 콘솔 사용 방법에 대한 자세한 내용은 X-Ray 콘솔 사용하기 섹션을 참조하십시오.

Amazon S3는 기본 http 헤더 계측을 지원합니다. X-Ray SDK는 AWS SDK를 통해 Amazon S3를 호출할 때 추적 헤더를 HTTP 헤더로 자동으로 채웁니다. 기본 추적 헤더는 `X-Amzn-Trace-Id`로 전달됩니다. 추적 헤더에 대한 자세한 내용은 개념 페이지의 추적 헤더를 참조하십시오. Amazon S3 추적 컨텍스트 전파는 다음 구독자를 지원합니다: Lambda, SQS 및 SNS. SQS와 SNS는 세그먼트 데이터를 자체적으로 전송하지 않기 때문에 트레이스 헤더를 다운스트림 서비스에 전파하더라도 S3에 의해 트리거될 때 트레이스 또는 트레이스 맵에 나타나지 않습니다.

Amazon S3 이벤트 알림 구성

Amazon S3 알림 기능을 사용하면 버킷에서 특정 이벤트가 발생하면 알림을 받을 수 있습니다. 이러한 알림은 애플리케이션 내에서 다음 대상에 전파할 수 있습니다:

- Amazon Simple Notification Service(Amazon SNS)
- Amazon Simple Queue Service(Amazon SQS)
- AWS Lambda

지원되는 이벤트 목록은 Amazon S3 개발자 가이드에서 지원되는 이벤트 유형을 참조하세요.

Amazon SNS 및 Amazon SQS

SNS 주제 또는 SQS 대기열에 알림을 게시하려면 먼저 Amazon S3 권한을 부여해야 합니다. 이러한 권한을 부여하려면 대상 SNS 주제 또는 SQS 대기열에 AWS Identity and Access Management (IAM) 정책을 연결합니다. 필요한 IAM 정책에 대해 자세히 알아보려면 SNS 주제 또는 SQS 대기열에 메시지를 게시할 권한 부여 섹션을 참조하십시오.

SNS와 SQS를 X-Ray와 통합하는 방법에 대한 자세한 내용은 Amazon SNS 및 AWS X-Ray 및 Amazon SQS 및 AWS X-Ray을 참조하십시오.

AWS Lambda

Amazon S3 콘솔을 사용하여 S3 버킷에 Lambda 함수용 이벤트 알림을 구성할 때 이 콘솔은 버킷에서 함수를 간접 호출하기 위한 권한을 Amazon S3에 부여하도록 Lambda 함수에 필수 권한을 설정합니다. 자세한 내용은 Amazon Simple Storage Service 콘솔 사용 설명서의 S3 버킷에 대한 이벤트 알림을 활성화하고 구성하려면 어떻게 해야 하나요?를 참조하세요.

에서 Lambda 함수를 호출 AWS Lambda 할 수 있는 Amazon S3 권한을 부여할 수도 있습니다. 자세한 내용은 AWS Lambda 개발자 안내서의 자습서: Amazon S3에서 Lambda 사용을 참조하세요. AWS

Lambda를 X-Ray와 통합하는 방법에 대한 자세한 내용은 AWS Lambda에서 Java 코드 계측을 참조하세요.

AWS Distro for OpenTelemetry 및 AWS X-Ray

AWS Distro for OpenTelemetry(ADOT)를 사용하여 지표 및 추적을 수집하여 Amazon CloudWatch, Amazon OpenSearch Service, Amazon Managed Service for Prometheus와 같은 AWS X-Ray 기타 모니터링 솔루션으로 전송합니다.

AWS OpenTelemetry용 배포판

AWS Distro for OpenTelemetry(ADOT)는 Cloud Native Computing Foundation(CNCF) OpenTelemetry 프로젝트를 기반으로 하는 AWS 배포입니다. OpenTelemetry는 분산된 트레이스 및 메트릭을 수집할 수 있는 단일 오픈 소스 API, 라이브러리 및 에이전트 세트를 제공합니다. 이 툴킷은 AWS에서 테스트, 최적화, 보안 및 지원하는 SDK, 자동 계측 에이전트 및 수집기를 포함한 업스트림 OpenTelemetry 구성 요소의 배포판입니다.

ADOT를 사용하면 엔지니어가 애플리케이션을 한 번 계측하고 상관관계가 있는 지표 및 추적을 Amazon CloudWatch, AWS X-Ray Amazon OpenSearch Service, Amazon Managed Service for Prometheus를 포함한 여러 AWS 모니터링 솔루션으로 전송할 수 있습니다.

ADOT는 점점 더 많은와 통합되어 X-Ray와 같은 모니터링 솔루션으로 추적 및 지표를 AWS 서비스 쉽게 전송할 수 있습니다. ADOT와 통합된 서비스의 몇 가지 예는 다음과 같습니다:

- AWS Lambda –ADOT용 AWS 관리형 Lambda 계층은 Lambda 함수를 자동으로 계측하고 OpenTelemetry를 설정하기 쉬운 계층에 AWS Lambda 및 X-Ray에 대한 out-of-the-box 구성과 함께 패키징하여 plug-and-play 사용자 경험을 제공합니다. 사용자는 코드를 변경하지 않고도 Lambda 함수에 대해 OpenTelemetry를 활성화 및 비활성화할 수 있습니다. 자세한 내용은 OpenTelemetry Lambda용AWS 배포판을 참조하세요.
- Amazon Elastic Container Service (ECS) — OpenTelemetry Collector용 AWS 배포판을 사용하여 Amazon ECS 애플리케이션에서 메트릭과 트레이스를 수집하여 X-Ray 및 기타 모니터링 솔루션으로 전송합니다. 자세한 내용은 Amazon ECS 개발자 안내서에서 애플리케이션 추적 데이터 수집을 참조하세요.
- AWS App Runner - App Runner는 AWS Distro for OpenTelemetry(ADOT)를 사용하여 X-Ray로 추적 전송을 지원합니다. ADOT SDK를 사용하여 컨테이너화된 애플리케이션에 대한 추적 데이터를 수집하고, X-Ray를 사용하여 계측된 애플리케이션을 분석하고 인사이트를 얻을 수 있습니다. 자세한 내용은 AWS App Runner 및 X-Ray를 참조하세요.

추가 와의 통합을 포함하여 AWS Distro for OpenTelemetry에 대한 자세한 내용은 AWS Distro for OpenTelemetry 설명서를 AWS 서비스참조하세요.

AWS Distro for OpenTelemetry 및 X-Ray를 사용하여 애플리케이션을 계측하는 방법에 대한 자세한 내용은 AWS Distro for OpenTelemetry를 사용하여 애플리케이션 계측을 참조하세요.

를 사용하여 X-Ray 암호화 구성 변경 사항 추적 AWS Config

AWS X-Ray 는와 통합되어 X-Ray 암호화 리소스에 대한 구성 변경을 AWS Config 기록합니다. AWS Config 를 사용하여 X-Ray 암호화 리소스를 인벤토리화하고, X-Ray 구성 기록을 감사하고, 리소스 변경 사항에 따라 알림을 보낼 수 있습니다.

AWS Config 는 다음과 같은 X-Ray 암호화 리소스 변경 사항을 이벤트로 로깅하도록 지원합니다.

- 구성 변경 – 암호화 키 변경 또는 추가, 또는 기본 X-Ray 암호화 설정으로 되돌리기.

다음 지침에 따라 X-Ray와 간에 기본 연결을 생성하는 방법을 알아봅니다 AWS Config.

Lambda 함수 트리거 생성

사용자 지정 AWS Config 규칙을 생성하려면 먼저 사용자 지정 AWS Lambda 함수의 ARN이 있어야 합니다. 다음 지침을 따라 Node.js를 사용하여 `XrayEncryptionConfig` 리소스의 상태를 기준으로 준수 또는 미준수 값을 AWS Config 로 반환하는 기본 함수를 생성합니다.

AWS::XrayEncryptionConfig 변경 트리거를 사용하여 Lambda 함수를 생성하려면

1. Lambda 콘솔을 엽니다. 함수 생성(Create function)을 선택합니다.
2. 블루프린트를 선택하고 블루프린트 라이브러리를 config-rule-change-triggered 블루프린트로 필터링합니다. 블루프린트의 이름에서 링크를 클릭하거나 구성을 선택하여 계속 진행합니다.
3. 다음 필드를 정의하여 블루프린트를 구성합니다.
   - [Name]에서 이름을 입력합니다.
   - [Role]에서 [Create new role from template(s)]를 선택합니다.
   - [Role name]에 이름을 입력합니다.
   - 정책 템플릿에서 AWS Config 규칙 권한을 선택합니다.
4. 함수 생성을 선택하여 AWS Lambda 콘솔에서 함수를 생성하고 표시합니다.
5. 함수 코드를 편집하여 `AWS::EC2::Instance`를 `AWS::XrayEncryptionConfig`로 바꿉니다. 설명 필드를 업데이트해서 이 변경 내용을 반영할 수도 있습니다.

   기본 코드

```
    if (configurationItem.resourceType !== 'AWS::EC2::Instance') {
        return 'NOT_APPLICABLE';
```

```
    } else if (ruleParameters.desiredInstanceType ===
configurationItem.configuration.instanceType) {
        return 'COMPLIANT';
    }
        return 'NON_COMPLIANT';
```

업데이트된 코드

```
    if (configurationItem.resourceType !== 'AWS::XRay::EncryptionConfig') {
        return 'NOT_APPLICABLE';
    } else if (ruleParameters.desiredInstanceType ===
configurationItem.configuration.instanceType) {
        return 'COMPLIANT';
    }
        return 'NON_COMPLIANT';
```

6. X-Ray에 액세스할 수 있도록 IAM 내 실행 역할에 다음을 추가합니다. 다음 권한은 X-Ray 리소스에 대한 읽기 전용 액세스를 허용합니다. 적절한 리소스에 대한 액세스를 제공하지 않으면 규칙과 연결된 Lambda 함수를 평가할 AWS Config 때에서 범위를 벗어납니다.

```
    {
        "Sid": "Stmt1529350291539",
        "Action": [
            "xray:GetEncryptionConfig"
        ],
        "Effect": "Allow",
        "Resource": "*"
     }
```

X-ray에 대한 사용자 지정 AWS Config 규칙 생성

Lambda 함수가 생성되면 함수의 ARN을 기록하고 AWS Config 콘솔로 이동하여 사용자 지정 규칙을 생성합니다.

X-Ray에 대한 AWS Config 규칙을 생성하려면

1. AWS Config 콘솔의 규칙 페이지를 엽니다.
2. 규칙 추가를 선택한 다음 사용자 지정 규칙 추가를 선택합니다.
3. AWS Lambda 함수 ARN에서 사용하려는 Lambda 함수와 연결된 ARN을 삽입합니다.

4. 설정할 트리거의 유형을 선택합니다.

   - 구성 변경 - 규칙의 범위와 일치하는 리소스가 구성에서 변경될 때 평가를 AWS Config 트리거합니다. 평가는가 구성 항목 변경 알림을 AWS Config 보낸 후에 실행됩니다.
   - 주기적 - 선택한 빈도(예: 24시간마다)로 규칙에 대한 평가를 AWS Config 실행합니다.

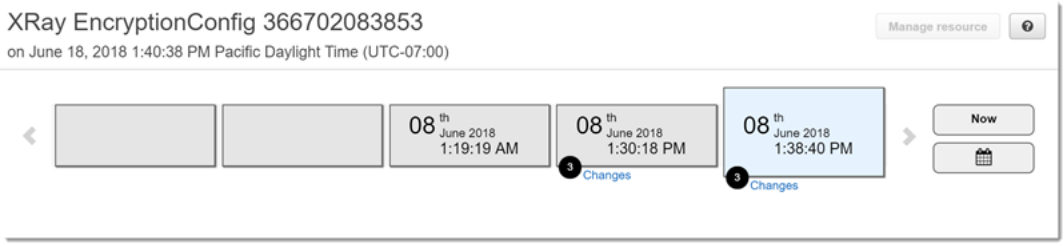
5. 리소스 유형으로는 X-Ray 섹션에서 EncryptionConfig를 선택합니다.
6. 저장(Save)을 선택합니다.

AWS Config 콘솔은 규칙의 규정 준수를 즉시 평가하기 시작합니다. 평가를 완료하는 데 몇 분 정도 걸릴 수 있습니다.

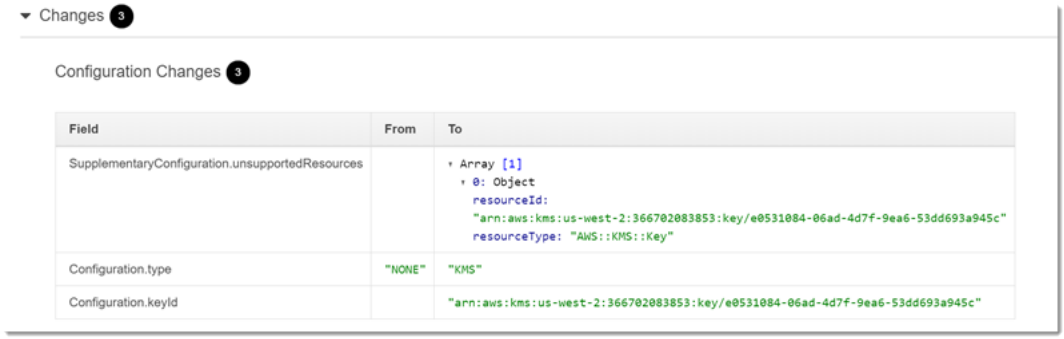
이제이 규칙이 준수되었으므로 감사 기록을 컴파일할 AWS Config 수 있습니다. AWS Config는 타임라인 형태로 리소스 변경 사항을 기록합니다. 이벤트 타임라인이 변경될 때마다는 암호화 키의 JSON 표현에서 변경된 내용을 표시하기 위해 시작/종료 형식으로 테이블을 AWS Config 생성합니다. EncryptionConfig와 연결된 2개의 필드 변경은 `Configuration.type` 및 `Configuration.keyID`입니다.

결과 예

다음은 특정 날짜 및 시간에 변경된 내용을 보여주는 AWS Config 타임라인의 예입니다.



다음은 AWS Config 변경 항목의 예입니다. 변경 내용이 전/후 형식으로 표시됩니다. 이 예는 기본 X-Ray 암호화 설정이 정의된 암호화 키로 변경된 것을 보여줍니다.



Amazon SNS 알림

구성 변경에 대한 알림을 받으려면 Amazon SNS 알림을 게시 AWS Config 하도록를 설정합니다. 자세한 내용은 이메일을 통한 AWS Config 리소스 변경 모니터링을 참조하세요.

AWS AppSync 그리고 AWS X-Ray

AWS AppSync에 대한 요청을 활성화하고 추적할 수 있습니다. 자세한 내용은 AWS X-Ray로 추적을 참조하세요.

AWS AppSync API에 대해 X-Ray 추적이 활성화되면 적절한 권한이 있는 AWS Identity and Access Management 서비스 연결 역할이 계정에 자동으로 생성됩니다. 이를 통해 AWS AppSync는 트레이스를 X-Ray에 안전한 방식으로 전송할 수 있습니다.

에 대한 Amazon API Gateway 활성 추적 지원 AWS X-Ray

사용자 요청이 Amazon API Gateway API를 통해 기본 서비스로 이동하는 동안 X-Ray를 사용하여 사용자 요청을 추적하고 분석할 수 있습니다. API Gateway는 모든 API Gateway 엔드포인트 유형(리전, 엣지 최적화, 프라이빗)에 대해 X-Ray 추적을 지원합니다. X-Ray를 사용할 수 AWS 리전 있는 모든 에서 Amazon API Gateway와 함께 X-Ray를 사용할 수 있습니다. 자세한 내용을 알아보려면 Amazon API Gateway 개발자 안내서의 AWS X-Ray를 사용한 API 게이트웨이 API 실행 추적을 참조하세요.

> **Note**
>
> X-Ray는 API 게이트웨이를 통한 REST API에 대한 추적만 지원합니다.

Amazon API Gateway는에 대한 활성 추적 지원을 제공합니다 AWS X-Ray. API 스테이지에서 활성 추적을 활성화하여 수신 요청을 샘플링하고 추적을 X-Ray에 전송합니다.

API 스테이지에서 활성 추적을 활성화하려면

1. https://console.aws.amazon.com/apigateway/에서 Amazon API Gateway 콘솔을 엽니다.
2. API를 선택합니다.
3. 스테이지를 선택합니다.
4. 로그/추적 탭에서 X-Ray Tracing 활성화를 선택한 다음 변경 사항 저장을 선택합니다.

5. 왼쪽 탐색 창에서 Resources(리소스)를 선택합니다.
6. 새로운 설정을 사용하여 API를 다시 배포하려면 Actions(작업) 드롭다운을 선택한 다음, Deploy API(API 배포)를 선택합니다.

API 게이트웨이는 X-Ray 콘솔에서 정의하는 샘플링 규칙을 사용하여 기록할 요청을 결정합니다. API에만 적용되는 규칙이나 특정 헤더가 포함된 요청에만 적용되는 규칙을 생성할 수 있습니다. API 게이트웨이는 단계 및 요청에 대한 세부 정보와 함께 세그먼트의 속성에 헤더를 기록합니다. 자세한 내용은 샘플링 규칙 구성 단원을 참조하십시오.

> **Note**
>
> API Gateway HTTP 통합을 통해 REST API를 트레이싱할 때 각 세그먼트의 서비스 이름은 API Gateway에서 HTTP 통합 엔드포인트로 향하는 요청 URL 경로로 설정되므로 각 고유 URL 경로에 대한 X-Ray 트레이스 맵에 서비스 노드가 생성됩니다. URL 경로가 많으면 트레이스 맵이 노드 한도인 10,000개를 초과하여 오류가 발생할 수 있습니다.
>
> API Gateway에서 생성하는 서비스 노드의 수를 최소화하려면 POST를 통해 URL 쿼리 문자열 또는 요청 본문에 파라미터를 전달하는 것을 고려해 보십시오. 어느 방법을 사용하든 파라미터가 URL 경로에 포함되지 않게 되므로 고유한 URL 경로와 서비스 노드 수가 줄어들 수 있습니다.

모든 수신 요청에 대해 API 게이트웨이는 추적 헤더가 아직 없는 수신 HTTP 요청에 추적 헤더를 추가합니다.

```
X-Amzn-Trace-Id: Root=1-5759e988-bd862e3fe1be46a994272793
```

X-Ray 트레이스 ID 형식

X-Ray `trace_id`는 하이픈으로 구분된 3개의 숫자로 구성됩니다. 예: `1-58406520-a006649127e371903a2de979`. 여기에는 다음이 포함됩니다.

- 버전 번호(1)
- 원래 요청의 시간(8자리 16진수를 사용하는 Unix 에포크 시간)

  예를 들어, 에포크 시간으로 2016년 12월 1일 오전 10시(PST)는 `1480615200`초이며, 16진수로는 `58406520`입니다.
- 트레이스의 96비트 글로벌 고유 식별자(24자리 16진수)

활성 추적이 비활성화된 경우에도 요청을 샘플링하고 추적을 시작한 서비스에서 요청이 오면 스테이지는 여전히 세그먼트를 기록합니다. 예를 들어, 구성된 웹 애플리케이션은 HTTP 클라이언트를 사용하여 API 게이트웨이 API를 직접 호출할 수 있습니다. X-Ray SDK를 사용하여 HTTP 클라이언트를 계측하면 샘플링 결정이 포함된 추적 헤더가 발신 요청에 추가됩니다. API 게이트웨이는 추적 헤더를 읽고 샘플링된 요청에 대한 세그먼트를 생성합니다.

API Gateway를 사용하여 API용 Java SDK를 생성하는 경우 SDK 클라이언트를 수동으로 계측하는 것과 동일한 방식으로 클라이언트 빌더에 요청 핸들러를 추가하여 AWS SDK 클라이언트를 계측할 수 있습니다. 자세한 내용은 Java용 X-Ray AWS SDK를 사용하여 SDK 호출 추적 섹션을 참조하세요.

Amazon EC2 및 AWS App Mesh

AWS X-Ray 는와 통합되어 마이크로서비스용 Envoy 프록시를 AWS App Mesh 관리합니다. App Mesh는 동일한 작업 또는 포드의 컨테이너에서 실행 중인 X-Ray 대몬(daemon)에 추적 데이터를 전송하도록 구성할 수 있는 Envoy 버전을 제공합니다. X-Ray는 다음과 같은 App Mesh 호환 서비스를 통한 추적을 지원합니다.

- Amazon Elastic Container Service(Amazon ECS)
- Amazon Elastic Kubernetes Service(Amazon EKS)
- Amazon Elastic Compute Cloud(Amazon EC2)

App Mesh를 통해 X-Ray 추적을 활성화하는 방법을 알아보려면 다음 지침을 사용합니다.

Envoy 프록시가 X-Ray로 데이터를 전송하도록 구성하려면 컨테이너 정의에서 `ENABLE_ENVOY_XRAY_TRACING` 환경 변수를 설정합니다.

> **Note**
>
> Envoy의 App Mesh 버전은 현재 구성된 샘플링 규칙을 기반으로 트레이스를 전송하지 않습니다. 대신 Envoy 버전 1.16.3 이상에서는 5% 의 고정 샘플링 속도를 사용하고, 1.16.3 이전의 Envoy 버전에는 50% 의 고정 샘플링 속도를 사용합니다.

Example Amazon ECS용 Envoy 컨테이너 정의

```
{
      "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**
>
> 사용 가능한 Envoy 지역 주소에 대한 자세한 내용은 AWS App Mesh 사용 설명서의 Envoy 이미지를 참조하십시오.

컨테이너에서 X-Ray 대몬(daemon)을 실행하는 방법에 대한 자세한 내용은 Amazon ECS에서 X-Ray 대몬(daemon) 실행하기 단원을 참조하십시오. 서비스 메시, 마이크로서비스, Envoy 프록시 및 X-Ray 대몬(daemon)이 포함된 샘플 애플리케이션의 경우 App Mesh Examples GitHub 리포지토리에 `colorapp` 샘플을 배포하십시오.

자세히 알아보기

- AWS App Mesh시작하기

- AWS App Mesh 및 Amazon ECS 시작하기

AWS App Runner 및 X-Ray

AWS App Runner는 소스 코드 또는 컨테이너 이미지에서의 확장 가능하고 안전한 웹 애플리케이션에 직접 배포할 수 있는 빠르고 간단하며 비용 효율적인 방법을 AWS 서비스 제공하는 입니다 AWS 클라우드. 새로운 기술을 배우거나, 사용할 컴퓨팅 서비스를 결정하거나, AWS 리소스를 프로비저닝하고 구성하는 방법을 알 필요가 없습니다. 자세한 내용은 AWS App Runner란 무엇입니까?를 참조하세요.

AWS App Runner는 AWS Distro for OpenTelemetry(ADOT)와 통합하여 트레이스를 X-Ray로 전송합니다. ADOT SDK를 사용하여 컨테이너화된 애플리케이션에 대한 추적 데이터를 수집하고, X-Ray를 사용하여 계측된 애플리케이션을 분석하고 인사이트를 얻을 수 있습니다. 자세한 내용은 X-Ray를 사용한 App Runner 애플리케이션 추적하기 을 참조하십시오.

를 사용하여 X-Ray API 호출 로깅 AWS CloudTrail

AWS X-Ray 는 사용자AWS CloudTrail, 역할 또는가 수행한 작업에 대한 레코드를 제공하는 서비스인와 통합됩니다 AWS 서비스. CloudTrail은 X-Ray에 대한 모든 API 직접 호출을 이벤트로 캡처합니다. 캡처되는 호출에는 X-Ray 콘솔에서 수행한 직접 호출과 X-Ray API 작업에 대한 코드 직접 호출이 포함됩니다. CloudTrail에서 수집한 정보를 사용하여 X-Ray에 대한 요청, 요청이 수행된 IP 주소, 요청이 수행된 시간, 추가 세부 정보를 확인할 수 있습니다.

모든 이벤트 또는 로그 항목에는 요청을 생성했던 사용자에 대한 정보가 포함됩니다. 자격 증명을 이용하면 다음을 쉽게 판단할 수 있습니다.

- 요청을 루트 사용자로 했는지 사용자 보안 인증으로 했는지 여부.
- IAM Identity Center 사용자를 대신하여 요청이 이루어졌는지 여부입니다.
- 역할 또는 페더레이션 사용자에 대한 임시 자격 증명을 사용하여 요청이 생성되었는지 여부.
- 다른 AWS 서비스에서 요청했는지 여부.

CloudTrail은 계정을 생성할 AWS 계정 때에서 활성화되며 CloudTrail 이벤트 기록에 자동으로 액세스할 수 있습니다. CloudTrail 이벤트 기록은 지난 90일 간 AWS 리전의 관리 이벤트에 대해 보기, 검색 및 다운로드가 가능하고, 수정이 불가능한 레코드를 제공합니다. 자세한 설명은 AWS CloudTrail 사용 설명서의 CloudTrail 이벤트 기록 작업을 참조하세요. Event history(이벤트 기록) 보기는 CloudTrail 요금이 부과되지 않습니다.

AWS 계정 지난 90일 동안 이벤트를 지속적으로 기록하려면 추적 또는 CloudTrail Lake 이벤트 데이터 스토어를 생성합니다.

CloudTrail 추적

CloudTrail은 추적을 사용하여 Amazon S3 버킷으로 로그 파일을 전송할 수 있습니다. 를 사용하여 생성된 모든 추적 AWS Management Console 은 다중 리전입니다. AWS CLI를 사용하여 단일 리전 또는 다중 리전 추적을 생성할 수 있습니다. 계정의 모든에서 활동을 캡처하므로 다중 리전 추적 AWS 리전 을 생성하는 것이 좋습니다. 단일 리전 추적을 생성하는 경우 추적의 AWS 리전에 로깅된 이벤트만 볼 수 있습니다. 추적에 대한 자세한 내용은 AWS CloudTrail 사용 설명서의 Creating a trail for your AWS 계정 및 Creating a trail for an organization을 참조하세요.

CloudTrail에서 추적을 생성하여 진행 중인 관리 이벤트의 사본 하나를 Amazon S3 버킷으로 무료로 전송할 수는 있지만, Amazon S3 스토리지 요금이 부과됩니다. CloudTrail 요금에 대한 자세한 내용은 AWS CloudTrail 요금을 참조하세요. Amazon S3 요금에 대한 자세한 내용은 Amazon S3 요금을 참조하세요.

CloudTrail Lake 이벤트 데이터 스토어

CloudTrail Lake를 사용하면 이벤트에 대해 SQL 기반 쿼리를 실행할 수 있습니다. CloudTrail Lake는 행 기반 JSON 형식의 기존 이벤트를 Apache ORC 형식으로 변환합니다. ORC는 빠른 데이터 검색에 최적화된 열 기반 스토리지 형식입니다. 이벤트는 이벤트 데이터 스토어로 집계되며, 이벤트 데이터 스토어는 고급 이벤트 선택기를 적용하여 선택한 기준을 기반으로 하는 변경 불가능한 이벤트 컬렉션입니다. 이벤트 데이터 스토어에 적용하는 선택기는 어떤 이벤트가 지속되고 쿼리할 수 있는지 제어합니다. CloudTrail Lake에 대한 자세한 내용은 AWS CloudTrail 사용 설명서의 AWS CloudTrail Lake 작업을 참조하세요.

CloudTrail Lake 이벤트 데이터 스토어 및 쿼리에는 비용이 발생합니다. 이벤트 데이터 스토어를 생성할 때 이벤트 데이터 스토어에 사용할 요금 옵션을 선택합니다. 요금 옵션에 따라 이벤트 모으기 및 저장 비용과 이벤트 데이터 스토어의 기본 및 최대 보존 기간이 결정됩니다. CloudTrail 요금에 대한 자세한 내용은 AWS CloudTrail 요금을 참조하세요.

주제

- CloudTrail의 X-Ray 관리 이벤트
- CloudTrail의 X-Ray 데이터 이벤트
- X-Ray 이벤트 예제

CloudTrail의 X-Ray 관리 이벤트

AWS X-Ray 는와 통합되어 X-Ray AWS 서비스 에서 사용자, 역할 또는가 수행한 API 작업을 AWS CloudTrail 기록합니다. CloudTrail을 사용하여 X-Ray API 요청을 실시간으로 모니터링하고 Amazon S3, Amazon CloudWatch Logs 및 Amazon CloudWatch Events에 로그를 저장할 수 있습니다. X-Ray 는 CloudTrail 로그 파일에 다음 작업을 이벤트로 로깅합니다.

지원되는 API 작업

- PutEncryptionConfig
- GetEncryptionConfig
- CreateGroup
- UpdateGroup
- DeleteGroup
- GetGroup
- GetGroups
- GetInsight
- GetInsightEvents
- GetInsightImpactGraph
- GetInsightSummaries
- GetSamplingStatisticSummaries

CloudTrail의 X-Ray 데이터 이벤트

데이터 이벤트는 리소스에 대해 또는 리소스에서 수행된 리소스 작업에 대한 정보를 제공합니다(예: 세그먼트 문서를 X-Ray에 업로드하는 PutTraceSegments).

이를 데이터 영역 작업이라고도 합니다. 데이터 이벤트가 대량 활동인 경우도 있습니다. 기본적으로 CloudTrail은 데이터 이벤트를 로깅하지 않습니다. CloudTrail 이벤트 기록은 데이터 이벤트를 기록하지 않습니다.

데이터 이벤트에는 추가 요금이 적용됩니다. CloudTrail 요금에 대한 자세한 내용은 AWS CloudTrail 요금을 참조하세요.

CloudTrail 콘솔 AWS CLI또는 CloudTrail API 작업을 사용하여 X-Ray 리소스 유형에 대한 데이터 이벤트를 로깅할 수 있습니다. 데이터 이벤트를 로깅하는 방법에 대한 자세한 내용은 AWS CloudTrail 사용

설명서의 Logging data events with the AWS Management Console 및 Logging data events with the AWS Command Line Interface를 참조하세요.

다음 표에는 데이터 이벤트를 로깅할 수 있는 X-Ray 리소스 유형이 나와 있습니다. 데이터 이벤트 유형(콘솔) 열에는 CloudTrail 콘솔의 데이터 이벤트 유형 목록에서 선택할 값이 표시됩니다. resources.type 값 열에는 AWS CLI 또는 CloudTrail APIs를 사용하여 고급 이벤트 선택기를 구성할 때 지정하는 `resources.type` 값이 표시됩니다. CloudTrail에 로깅되는 데이터 API 열에는 리소스 유형에 대해 CloudTrail에 로깅된 API 호출이 표시됩니다.

데이터 이벤트 유형(콘솔)	resources.type 값	CloudTrail에 로깅되는 데이터 API
X-Ray 트레이스	`AWS::XRay::Trace`	• PutTraceSegments • GetTraceSummaries • GetTraceGraph • GetServiceGraph • BatchGetTraces • GetTimeSeriesServiceStatistics • PutTelemetryRecords • GetSamplingTargets

`eventName` 및 `readOnly` 필드를 필터링하여 중요한 이벤트만 로깅하도록 고급 이벤트 선택기를 구성할 수 있습니다. 그러나 X-Ray 트레이스에 ARN이 없기 때문에 `resources.ARN` 필드 선택기를 추가하여 이벤트를 선택할 수 없습니다. 이러한 필드에 대한 자세한 내용은 AWS CloudTrail API 참조의 AdvancedFieldSelector 섹션을 참조하세요. 다음은 `put-event-selectors` AWS CLI 명령을 실행하여 CloudTrail 추적에서 데이터 이벤트를 로깅하는 방법의 예입니다. 명령을 실행하거나 추적이 생성된 리전을 지정해야 합니다. 그렇지 않으면 작업에서 `InvalidHomeRegionException` 예외가 반환됩니다.

```
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 이벤트 예제

관리 이벤트 예제, **GetEncryptionConfig**

다음은 CloudTrail의 X-Ray GetEncryptionConfig 로그 항목에 대한 예제입니다.

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"
}
```

데이터 이벤트 예제, **PutTraceSegments**

다음은 CloudTrail의 X-Ray PutTraceSegments 데이터 이벤트 로그 항목에 대한 예제입니다.

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"
  }
}
```


CloudWatch와 X-Ray 통합

AWS X-Ray 는 CloudWatch Application Signals, CloudWatch RUM 및 CloudWatch Synthetics와 통합되어 애플리케이션의 상태를 더 쉽게 모니터링할 수 있습니다. Application Signals용 애플리케이션을 활성화하여 서비스, 클라이언트 페이지, Synthetics Canary 및 서비스 종속성의 운영 상태를 모니터링하고 문제를 해결할 수 있습니다.

CloudWatch 지표, 로그, X-Ray 트레이스의 상관 관계를 통해 X-Ray 트레이스 맵은 서비스에 대한 전체적인 보기를 제공하여 성능 병목 현상을 신속하게 찾아내고 영향을 받는 사용자를 식별할 수 있도록 도와줍니다.

CloudWatch RUM을 사용하면 실제 사용자 모니터링을 수행하여 실제 사용자 세션에서 웹 애플리케이션 성능에 대한 클라이언트 측 데이터를 거의 실시간으로 수집하고 볼 수 있습니다. AWS X-Ray 및 CloudWatch RUM을 사용하면 애플리케이션의 최종 사용자부터 다운스트림 AWS 관리형 서비스까지 요청 경로를 분석하고 디버깅할 수 있습니다. 이를 통해 최종 사용자에게 영향을 미치는 지연 시간 추세와 오류를 파악할 수 있습니다.

주제

- CloudWatch RUM 및 AWS X-Ray
- X-Ray를 사용하여 CloudWatch Synthetics canary 디버깅하기

CloudWatch RUM 및 AWS X-Ray

Amazon CloudWatch RUM을 사용하면 실제 사용자 모니터링을 수행하여 실제 사용자 세션에서 웹 애플리케이션 성능에 대한 클라이언트 측 데이터를 거의 실시간으로 수집하고 볼 수 있습니다. AWS X-Ray 및 CloudWatch RUM을 사용하면 애플리케이션의 최종 사용자부터 다운스트림 AWS 관리형 서비스를 통해 요청 경로를 분석하고 디버깅할 수 있습니다. 이를 통해 최종 사용자에게 영향을 미치는 지연 시간 추세와 오류를 파악할 수 있습니다.

사용자 세션에 대한 X-Ray 추적을 활성화하면 CloudWatch RUM은 허용된 HTTP 요청에 X-Ray 추적 헤더를 추가하고, 허용된 HTTP 요청에 대한 X-Ray 세그먼트를 기록합니다. 그런 다음 이러한 사용자 세션의 트레이스와 세그먼트를 X-Ray 및 CloudWatch 콘솔에서 X-Ray 트레이스 맵을 포함하여 확인할 수 있습니다.

> **Note**
>
> CloudWatch RUM은 X-Ray 샘플링 규칙과 통합되지 않습니다. 대신 CloudWatch RUM을 사용하도록 애플리케이션을 설정할 때 샘플링 비율을 선택하세요. CloudWatch RUM에서 전송된 트레이스는 추가 비용이 발생할 수 있습니다. 자세한 내용은 AWS X-Ray 요금을 참조하세요.

기본적으로 CloudWatch RUM에서 전송된 클라이언트 측 추적은 서버 측 추적에 연결되지 않습니다. 클라이언트 측 추적을 서버 측 추적에 연결하려면 이러한 HTTP 요청에 X-Ray 추적 헤더를 추가하도록 CloudWatch RUM 웹 클라이언트를 구성합니다.

> **Warning**
>
> HTTP 요청에 X-Ray 추적 헤더를 추가하도록 CloudWatch RUM 웹 클라이언트를 구성하면 교차 출처 리소스 공유(CORS)가 실패할 수 있습니다. 이를 방지하려면 다운스트림 서비스의 CORS 구성에서 허용되는 헤더 목록에 `X-Amzn-Trace-Id` HTTP 헤더를 추가하세요. API Gateway를 다운스트림으로 사용하는 경우 REST API 리소스에 대한 CORS 활성화를 참조하십시오. 프로덕션 환경에서 클라이언트 측 X-Ray 추적 헤더를 추가하기 전에 애플리케이션을 테스트하는 것을 강력하게 권장합니다. 자세한 내용은 CloudWatch RUM 웹 클라이언트 설명서를 참조하세요.

CloudWatch의 실제 사용자 모니터링에 대한 자세한 내용은 CloudWatch RUM 사용을 참조하십시오. X-Ray로 사용자 세션 추적을 포함하여 CloudWatch RUM을 사용하도록 애플리케이션을 설정하려면 CloudWatch RUM을 사용하도록 애플리케이션 설정을 참조하십시오.

X-Ray를 사용하여 CloudWatch Synthetics canary 디버깅하기

CloudWatch Synthetices는 하루 24시간 분당 1회 실행되는 스크립팅된 canary를 사용하여 엔드포인트와 API를 모니터링할 수 있는 완전 관리형 서비스입니다.

카나리아 스크립트를 사용자 지정하여 다음 항목의 변경 사항을 확인할 수 있습니다.

- 가용성
- 지연 시간
- 트랜잭션
- 손상되거나 연결이 끊어진 링크
- 단계별 작업 완료
- 페이지 로드 오류
- UI 자산의 로드 지연 시간
- 복잡한 마법사 흐름
- 애플리케이션의 체크아웃 흐름

카나리아는 동일한 경로를 따르고, 고객과 동일한 작업과 동작을 수행하고, 고객 경험을 지속적으로 확인합니다.

Synthetics 테스트 설정에 대한 자세한 내용은 Synthetics를 사용하여 카나리아 생성 및 관리를 참조하십시오.

다음 예제는 Synthetics 카나리아가 제기하는 문제를 디버깅하기 위한 일반적인 사용 사례를 보여줍니다. 각 예제는 트레이스 맵 또는 X-Ray Analytics 콘솔을 사용하여 디버깅하기 위한 주요 전략을 보여줍니다.

트레이스 맵을 읽는 방법과 트레이스 맵과 상호 작용하는 자세한 방법은 서비스 맵 보기 단원을 참조하세요.

X-Ray Analytics 콘솔을 읽고 상호 작용하는 방법에 대한 자세한 내용은 AWS X-Ray Analytics 콘솔과 상호 작용을 참조하세요.

주제

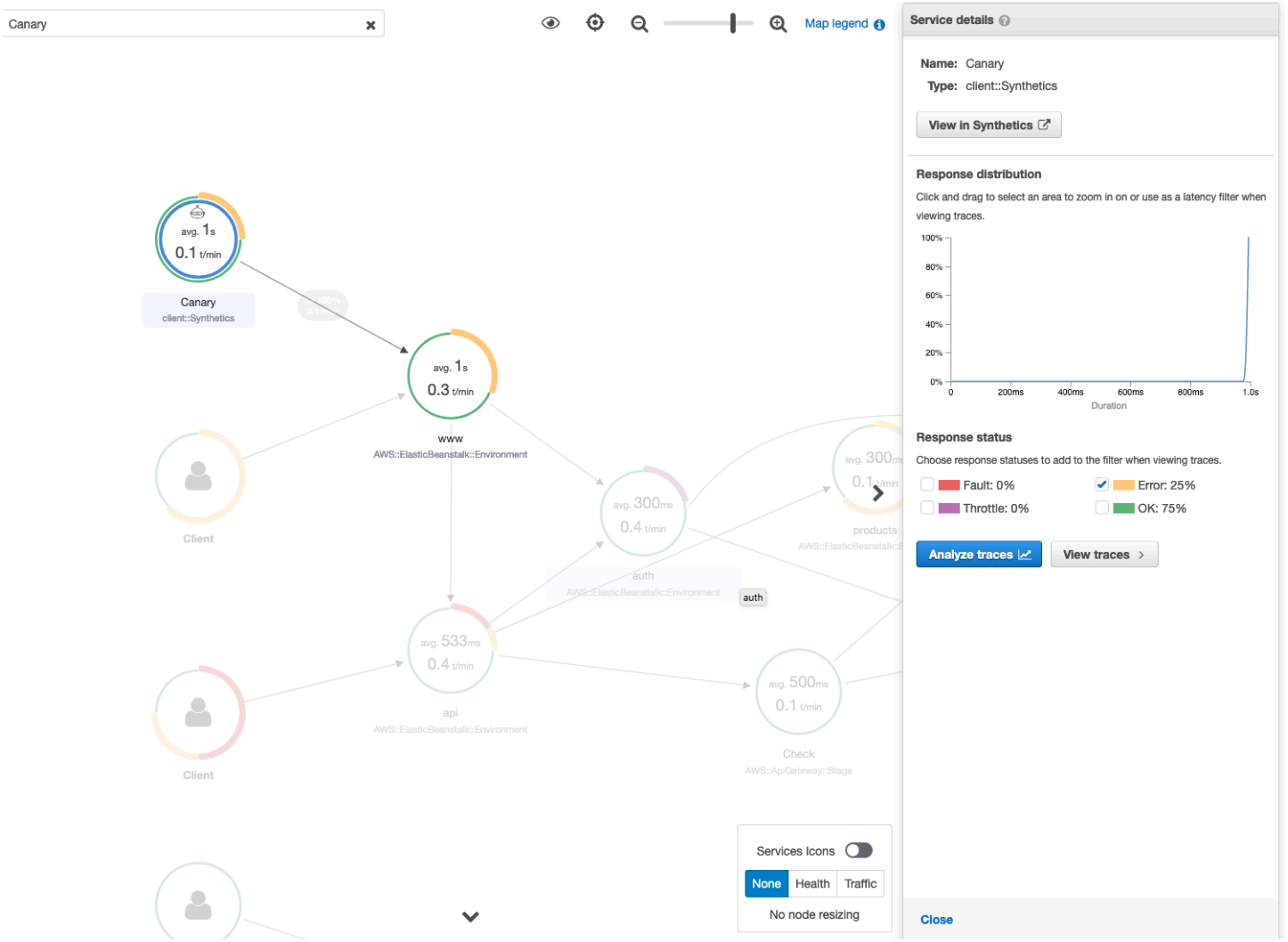
- 서비스 맵에서 오류 보고가 증가한 카나리 보기
- 개별 트레이스에 대한 트레이스 세부 정보 맵을 사용하여 각 요청의 세부 정보 보기

- 업스트림 및 다운스트림 서비스에서 지속적인 실패의 근본 원인 파악
- 성능 병목 현상 및 추세 파악
- 변경 전후의 지연 시간과 오류율 또는 장애율 비교
- 모든 API 및 URL에 필요한 카나리아 적용 범위 결정
- Synthetics 테스트에 중점을 둔 그룹 사용

서비스 맵에서 오류 보고가 증가한 카나리 보기

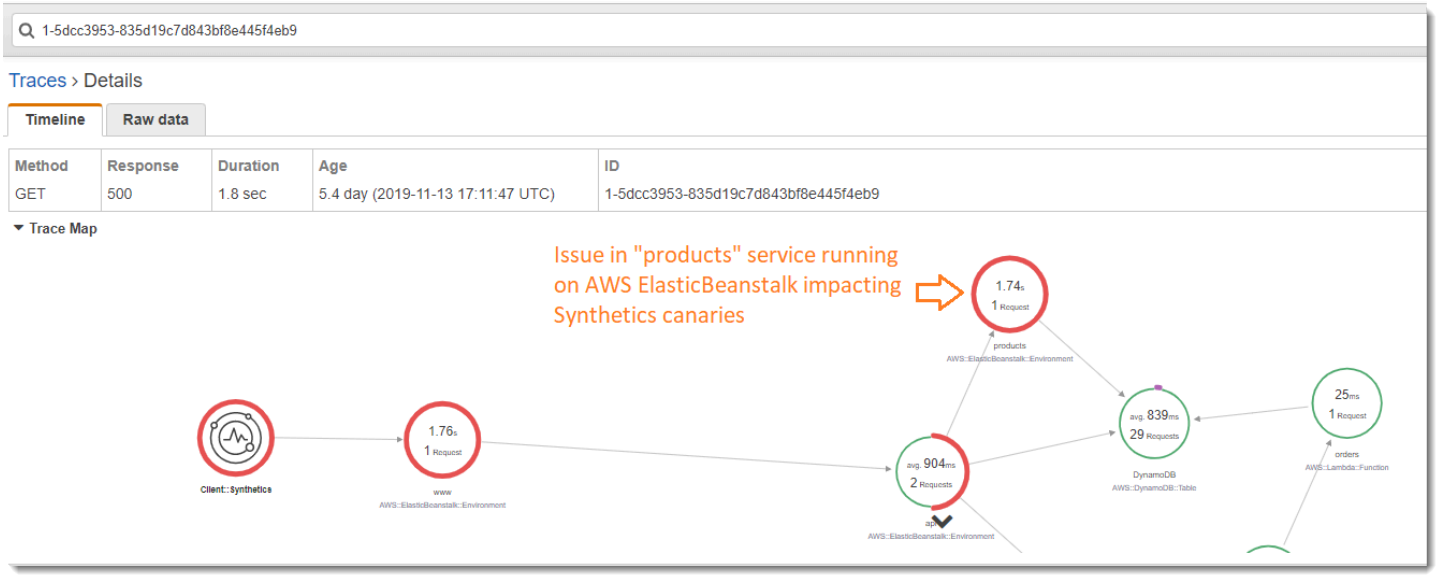
X-Ray 트레이스 맵 내에서 오류, 장애, 스로틀링 속도 또는 느린 응답 시간이 증가한 카나리를 확인하려면 `Client::Synthetic` 필터를 사용하여 Synthetics Canary 클라이언트 노드를 강조 표시할 수 있습니다. 노드를 클릭하면 전체 요청의 응답 시간 분포가 표시됩니다. 두 노드 사이의 엣지를 클릭하면 해당 연결을 통과한 요청에 대한 세부 정보가 표시됩니다. 트레이스 맵에서 관련 다운스트림 서비스에 대해 '원격' 추론된 노드를 볼 수도 있습니다.

Synthetics 노드를 클릭하면 사이드 패널에 Synthetics에서 보기 버튼이 있으며, 이 버튼을 클릭하면 Synthetics 콘솔로 리디렉션되어 canary 세부 정보를 확인할 수 있습니다.



개별 트레이스에 대한 트레이스 세부 정보 맵을 사용하여 각 요청의 세부 정보 보기

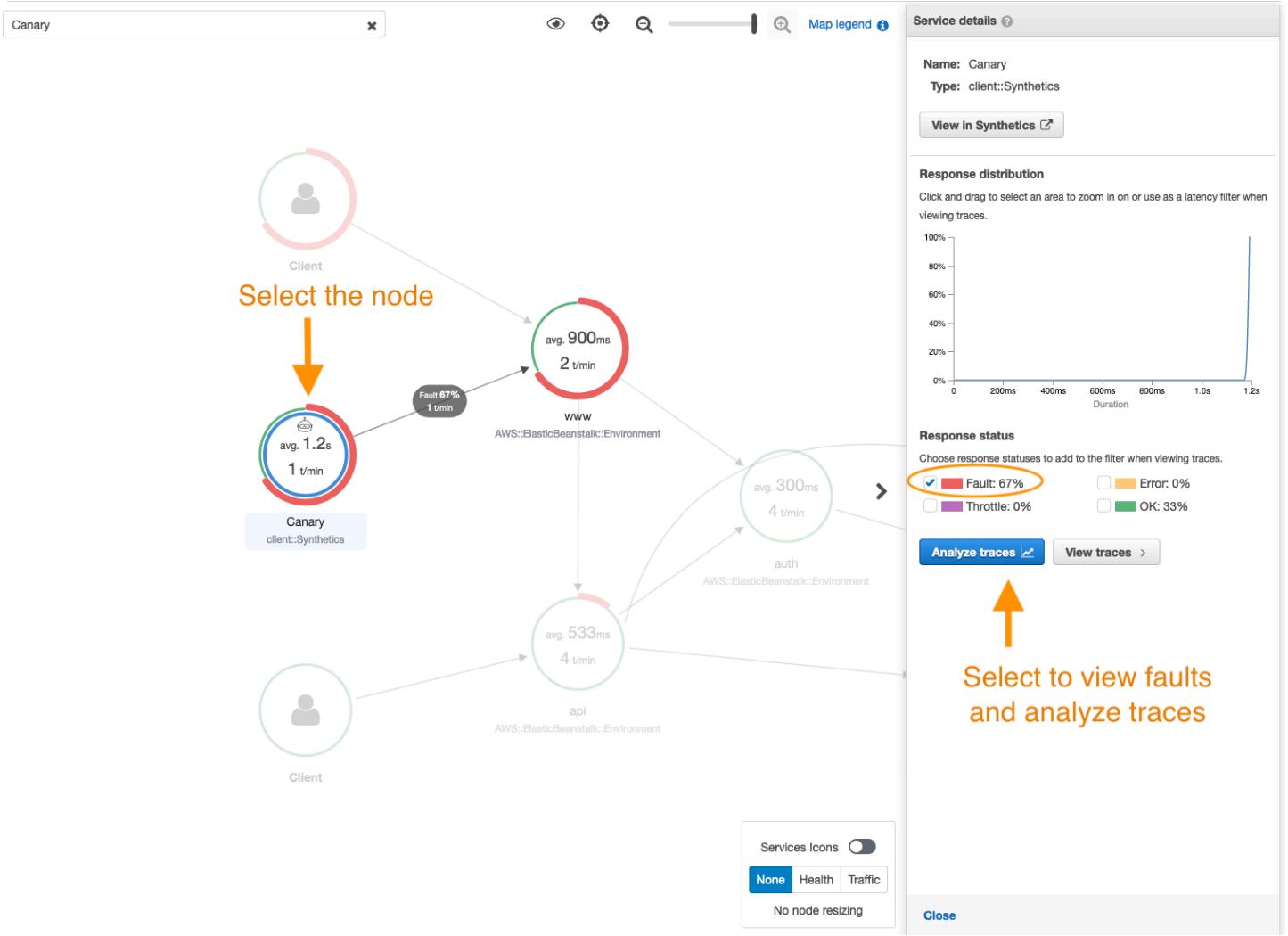
지연 시간이 가장 길거나 오류를 일으키는 서비스를 확인하려면 트레이스 맵에서 트레이스를 선택하여 트레이스 세부 정보 맵을 간접 호출합니다. 개별 트레이스 세부 정보 맵에는 단일 요청의 종단 간 경로가 표시됩니다. 이 정보를 사용하여 호출된 서비스를 파악하고 업스트림 및 다운스트림 서비스를 시각화할 수 있습니다.

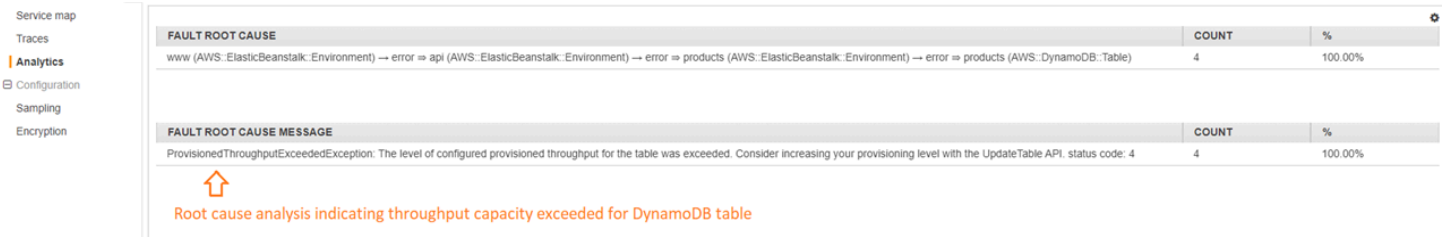


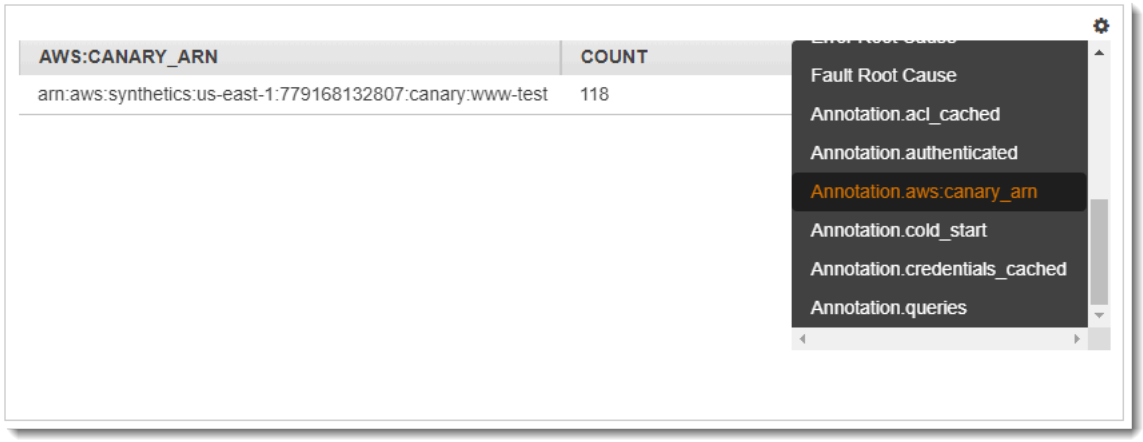
업스트림 및 다운스트림 서비스에서 지속적인 실패의 근본 원인 파악

Synthetics canary의 실패에 대한 CloudWatch 경보를 받으면 X-Ray의 트레이스 데이터에 대한 통계 모델링을 사용하여 X-Ray Analytics 콘솔 내에서 해당 문제에 대해 가능성 있는 근본 원인을 파악합니다. Analytics 콘솔의 응답 시간 근본 원인 테이블에는 기록된 개체 경로가 표시됩니다. X-Ray는 트레이스에서 응답 시간 원인으로 가장 가능성이 높은 경로를 결정합니다. 포맷은 발생하는 개체 계층 구조를 나타내며, 응답 시간 근본 원인으로 끝납니다.

다음 예제에서는 Amazon DynamoDB 테이블의 처리량 용량 예외로 인해 API 게이트웨이에서 실행되는 API “XXX”에 대한 Synthetics 테스트가 실패함을 보여줍니다.

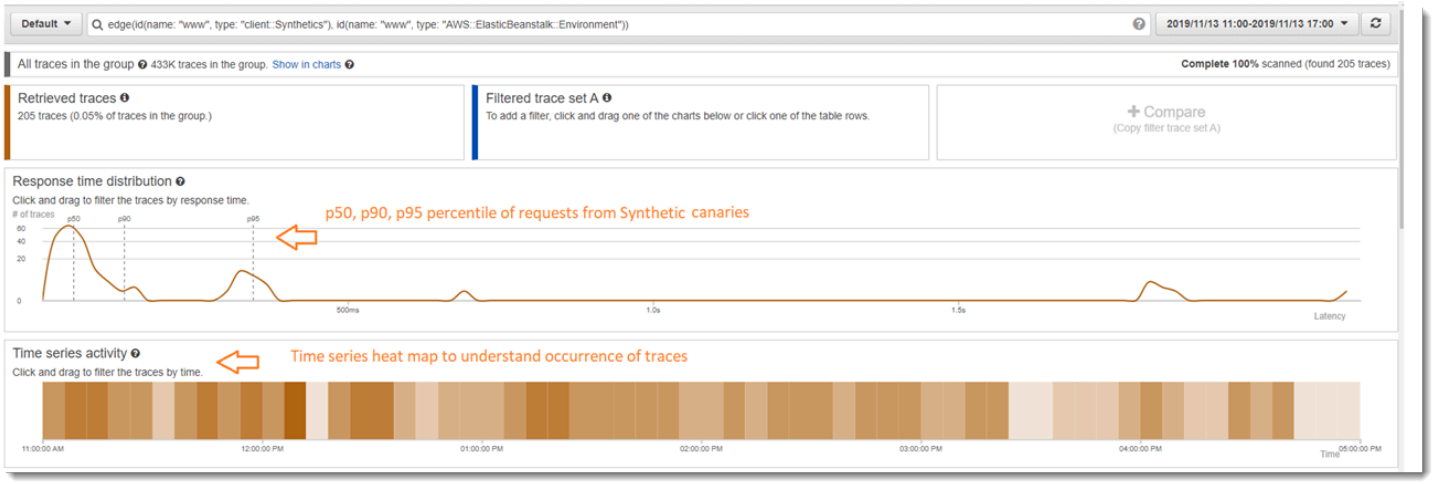

Canary
Map legend
Service details
Name: Canary
Type: client::Synthetics
View in Synthetics
Response distribution
Click and drag to select an area to zoom in on or use as a latency filter when viewing traces.
100%
80%
60%
40%
20%
0%
0
200ms
400ms
600ms
800ms
1.0s
1.2s
Duration
Response status
Choose response 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



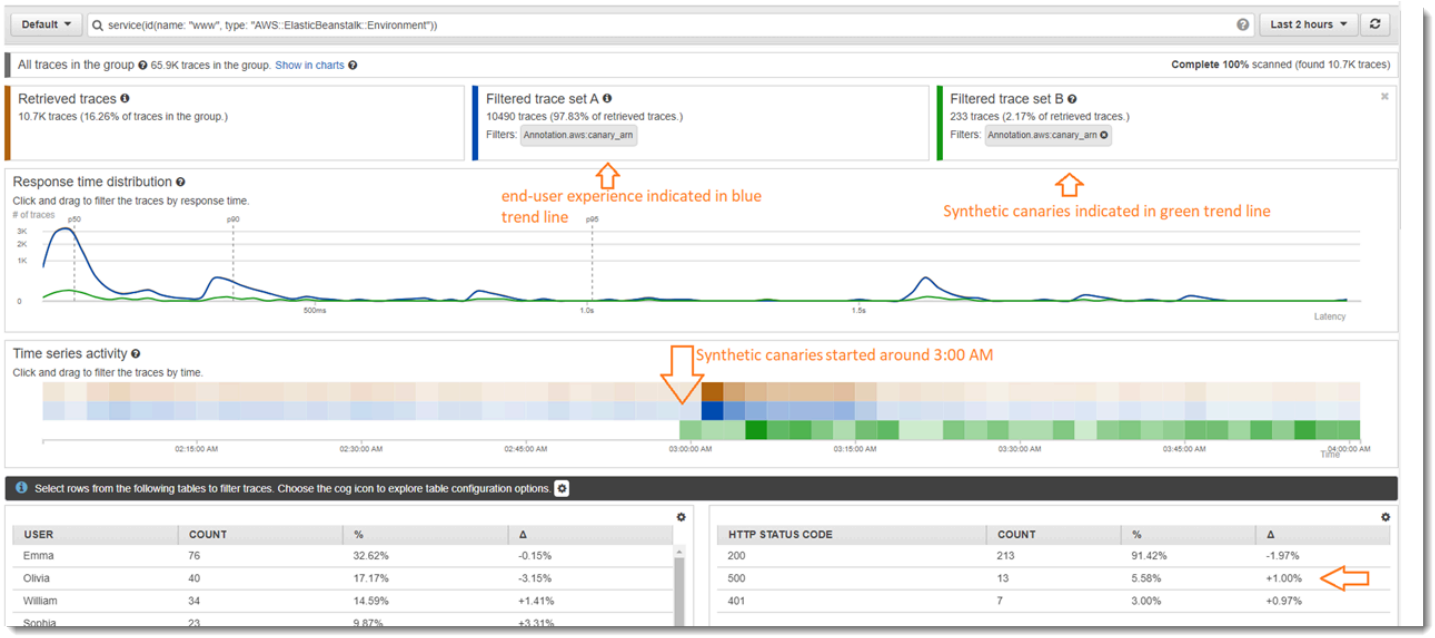
성능 병목 현상 및 추세 파악

일정 기간 동안 트레이스 세부 정보 맵을 채우는 Synthetics 카나리의 지속적인 트래픽을 사용하여 시간 경과에 따른 엔드포인트 성능의 추세를 확인할 수 있습니다.



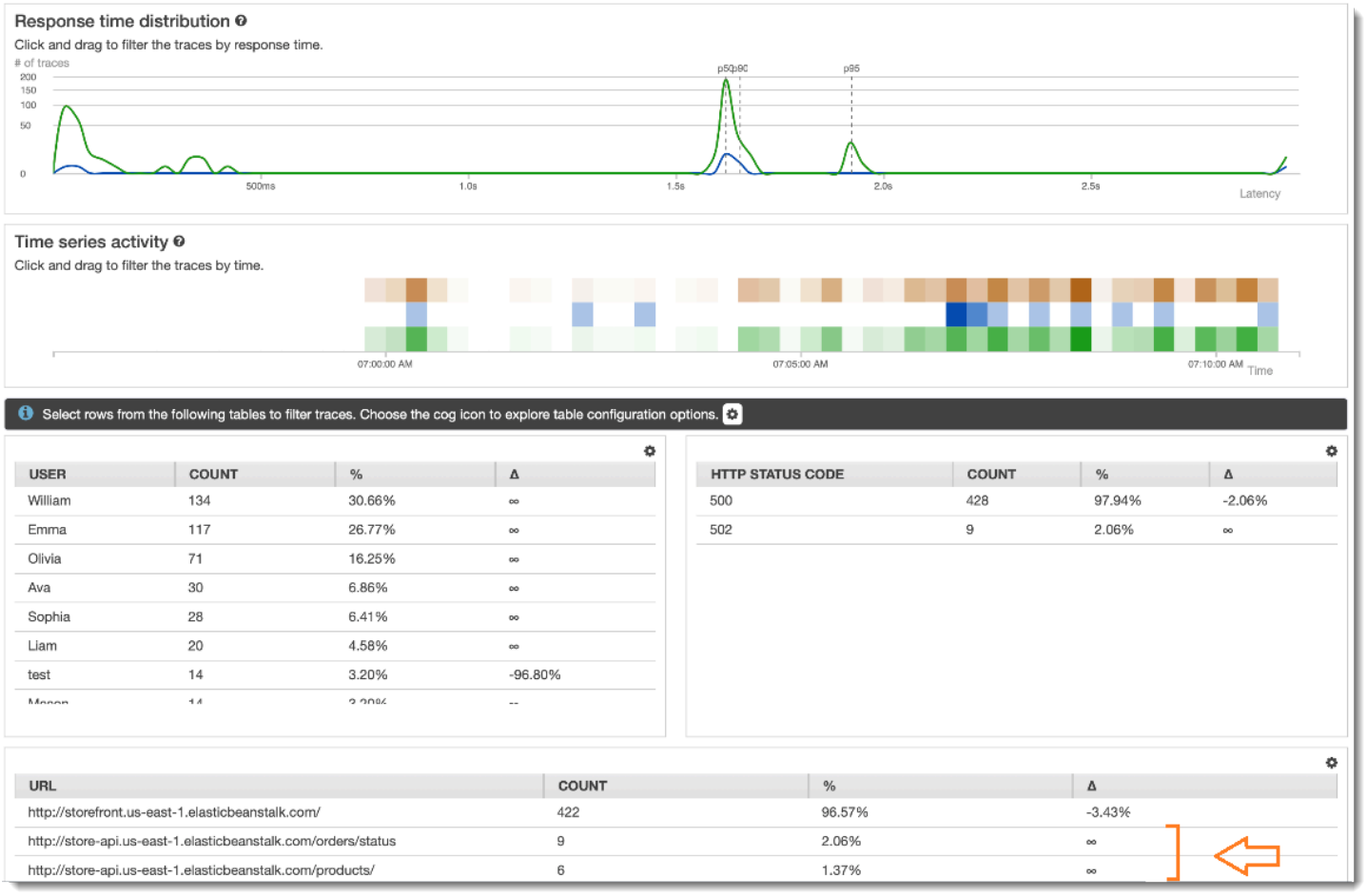
변경 전후의 지연 시간과 오류율 또는 장애율 비교

변경이 발생한 시간을 정확히 파악하여 카나리에서 발견한 문제의 증가와 해당 변경을 연관시킬 수 있습니다. X-Ray Analytics 콘솔을 사용하여 이전 및 이후 시간 범위를 다른 트레이스 세트로 정의하여 응답 시간 분포에서 시각적으로 구분할 수 있도록 만듭니다.



모든 API 및 URL에 필요한 카나리아 적용 범위 결정

X-Ray Analytics를 사용하여 카나리아의 경험을 사용자와 비교합니다. 다음 UI는 카나리아에 대한 파란색 추세선과 사용자에 대한 녹색 추세선을 보여줍니다. 또한 세 개의 URL 중 두 개에 카나리아 테스트가 없다는 것을 알 수 있습니다.

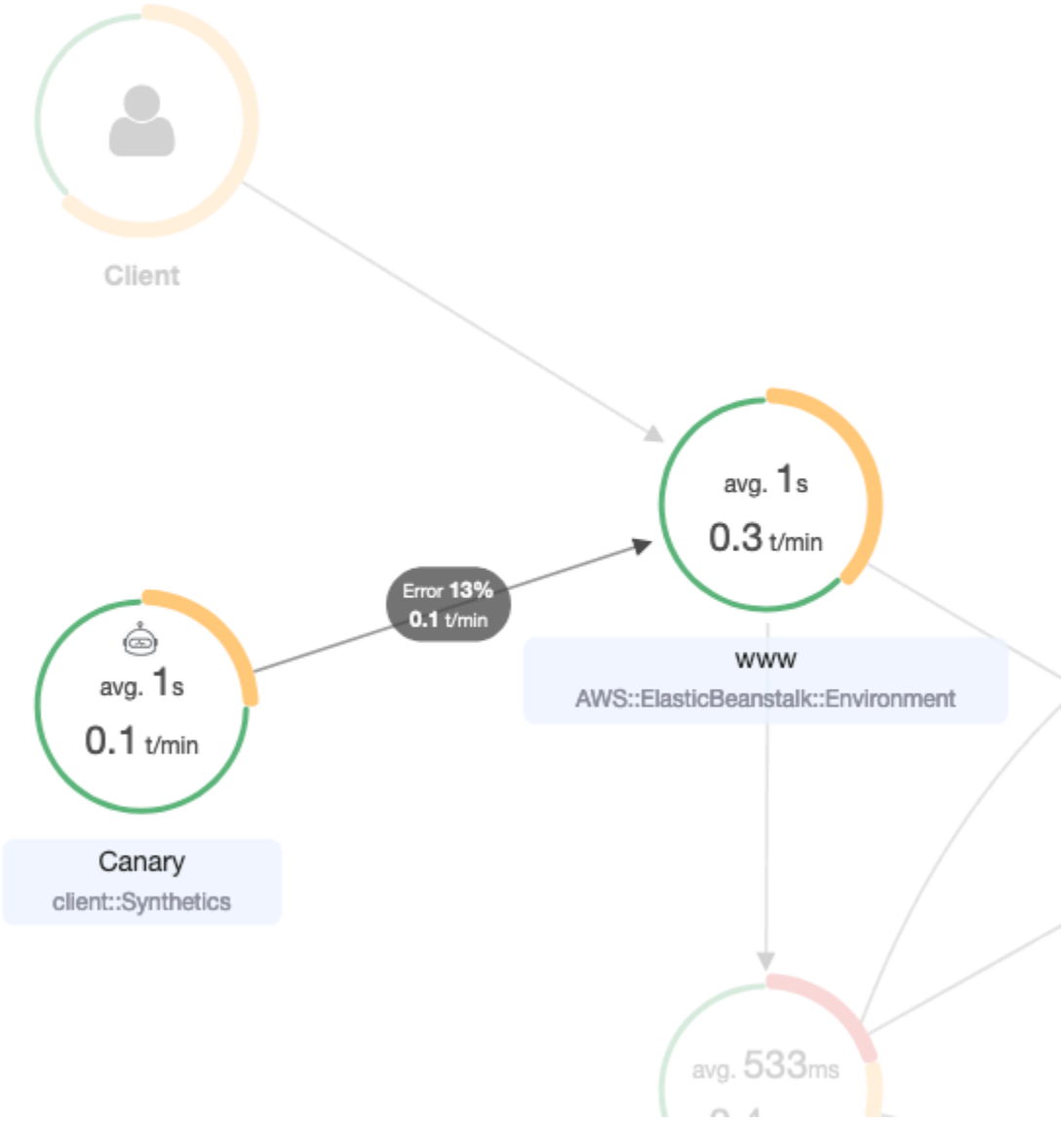


Synthetics 테스트에 중점을 둔 그룹 사용

필터 표현식을 사용하여 X-Ray 그룹을 만들어 특정 워크플로 집합(예: AWS Elastic Beanstalk에서 실행되는 애플리케이션 'www'에 대한 합성 테스트)에 집중할 수 있습니다. 복잡한 키워드 `service()`와 `edge()`을 사용하여 서비스 및 엣지를 필터링할 수 있습니다.

Example 그룹 필터 표현식

```
"edge(id(name: "www", type: "client::Synthetics"), id(name: "www", type:
 "AWS::ElasticBeanstalk::Environment"))"
```



AWS Elastic Beanstalk 그리고 AWS X-Ray

AWS Elastic Beanstalk 플랫폼에는 X-Ray 데몬이 포함됩니다. Elastic Beanstalk 콘솔에서 옵션을 설정하거나 구성 파일을 사용하여 대몬(daemon)을 실행할 수 있습니다.

Java SE 플랫폼에서는 Buildfile 파일을 사용하면 인스턴스 상에서 Maven 또는 Gradle로 애플리케이션을 빌드할 수 있습니다. Java용 X-Ray SDK 및 AWS SDK for Java 는 Maven에서 사용할 수 있으므로 애플리케이션 코드만 배포하고 인스턴스를 구축하여 모든 종속성을 번들링하고 업로드하지 않아도 됩니다.

Elastic Beanstalk 환경 속성을 사용하여 X-Ray SDK를 구성할 수 있습니다. Elastic Beanstalk가 환경 속성을 애플리케이션에 전달하는 데 사용하는 방법은 플랫폼마다 다릅니다. 플랫폼에 따라 X-Ray SDK의 환경 변수 또는 시스템 속성을 사용하십시오.

- Node.js 플랫폼 — 환경 변수 사용
- Java SE변수 플랫폼 — 환경 변수 사용
- Tomcat 플랫폼 — 시스템 속성 사용

자세한 내용은 AWS Elastic Beanstalk 개발자 안내서의 AWS X-Ray 디버깅 구성을 참조하세요.

Elastic Load Balancing 및 AWS X-Ray

Elastic Load Balancing 애플리케이션 로드 밸런서가 `X-Amzn-Trace-Id`라는 헤더에서 수신 HTTP 요청에 트레이스 ID를 추가합니다.

```
X-Amzn-Trace-Id: Root=1-5759e988-bd862e3fe1be46a994272793
```

X-Ray 트레이스 ID 형식

X-Ray `trace_id`는 하이픈으로 구분된 3개의 숫자로 구성됩니다. 예: `1-58406520-a006649127e371903a2de979`. 여기에는 다음이 포함됩니다.

- 버전 번호(1)
- 원래 요청의 시간(8자리 16진수를 사용하는 Unix 에포크 시간)

  예를 들어, 에포크 시간으로 2016년 12월 1일 오전 10시(PST)는 `1480615200`초이며, 16진수로는 `58406520`입니다.
- 트레이스의 96비트 글로벌 고유 식별자(24자리 16진수)

로드 밸런서는 X-Ray로 데이터를 전송하지 않으며 서비스 맵에 노드로 표시되지 않습니다.

자세한 내용은 Elastic Load Balancing 개발자 가이드에서 Application Load Balancer에 대한 추적 요청을 참조하세요.

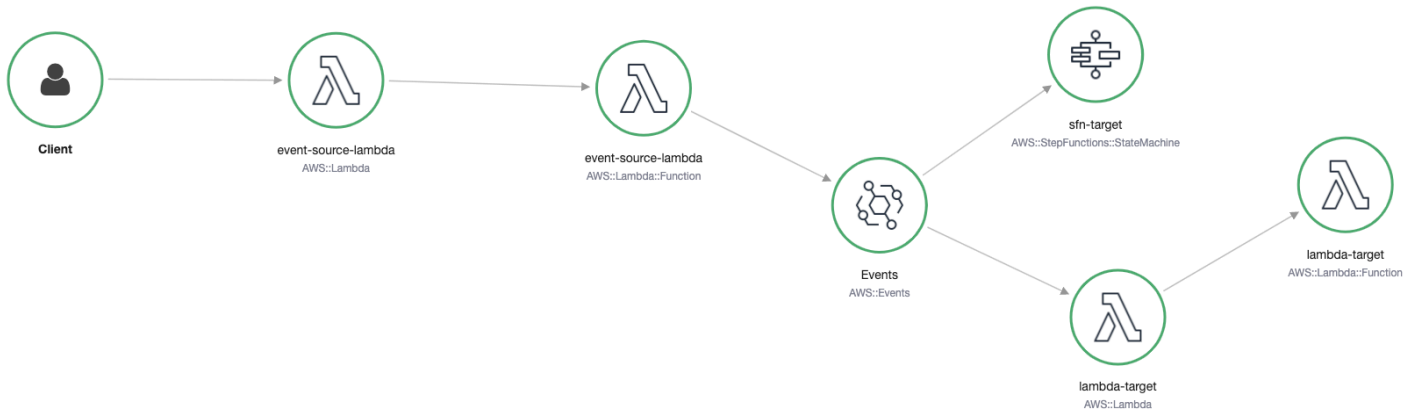
Amazon EventBridge 및 AWS X-Ray

AWS X-Ray 는 Amazon EventBridge와 통합되어 EventBridge를 통해 전달되는 이벤트를 추적합니다. X-Ray SDK로 계측하는 서비스가 이벤트를 EventBridge로 전송하면 추적 컨텍스트가 추적 헤더 내의 다운스트림 이벤트 타겟으로 전달됩니다. X-Ray SDK는 자동으로 추적 헤더를 가져와 이후의 모든 계측에 적용합니다. 이러한 연속성을 통해 사용자는 다운스트림 서비스 전반을 추적, 분석, 디버깅할 수 있으며 시스템을 보다 완벽하게 파악할 수 있습니다.

자세한 내용은 Amazon EventBridge 사용 설명서의 EventBridge X-Ray 통합을 참조하세요.

X-Ray 서비스 맵에서 소스 및 대상 보기

X-Ray 트레이스 맵에는 다음 예와 같이 소스 및 대상 서비스를 연결하는 EventBridge 이벤트 노드가 표시됩니다.



추적 컨텍스트를 이벤트 대상으로 전파하기

X-Ray SDK를 사용하면 EventBridge 이벤트 소스가 추적 컨텍스트를 다운스트림 이벤트 대상으로 전파할 수 있습니다. 다음 언어별 예제는 활성 추적이 활성화된 Lambda 함수에서 EventBridge를 호출하는 방법을 보여줍니다.

Java

X-Ray에 필요한 종속성을 추가하세요.

- AWS X-Ray SDK for Java
- AWS X-Ray Java용 레코더 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

requirements.txt 파일에 다음 종속성을 추가하세요.

```
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#

C# 종속성에 다음 X-Ray 패키지를 추가하세요.

```
<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 그리고 AWS X-Ray

AWS X-Ray 를 사용하여 AWS Lambda 함수를 추적할 수 있습니다. Lambda는 X-Ray 대몬(daemon)을 실행하고 함수 호출 및 실행에 대한 세부 정보가 포함된 세그먼트를 기록합니다. 추가 구

성을 위해 사용자 함수와 함께 X-Ray SDK를 번들링하여 발신 호출을 기록하고 주석 및 메타데이터를 추가할 수 있습니다.

다른 구성된 서비스에서 Lambda 함수를 호출하는 경우 는 이미 샘플링된 요청을 추가 구성 없이 추적합니다. 업스트림 서비스는 구성된 웹 애플리케이션 또는 다른 Lambda 함수일 수 있습니다. 서비스는 계측된 AWS SDK 클라이언트로 직접 또는 계측된 HTTP 클라이언트로 API Gateway API를 호출하여 함수를 호출할 수 있습니다.

AWS X-Ray 는 AWS Lambda 및 Amazon SQS를 사용하여 이벤트 기반 애플리케이션 추적을 지원합니다. CloudWatch 콘솔을 사용하면 각 요청이 Amazon SQS로 대기열에 추가되고 다운스트림 Lambda 함수에 의해 처리될 때 연결된 보기를 볼 수 있습니다. 업스트림 메시지 생산자의 트레이스가 다운스트림 Lambda 함수의 트레이스에 자동으로 연결되므로 전체 애플리케이션을 종합적으로 파악할 수 있습니다. 자세한 내용은 이벤트 중심 애플리케이션 추적을 참조하세요.

> **Note**
>
> 다운스트림 Lambda 함수에 대해 트레이스가 활성화된 경우, 다운스트림 함수가 트레이스를 생성하려면 다운스트림 함수를 직접 호출하는 루트 Lambda 함수에 대해서도 트레이스가 활성화되어 있어야 합니다.

Lambda 함수가 예정대로 실행되거나 구성되지 않은 서비스에서 호출되는 경우 활성 추적을 통해 호출을 샘플링하고 기록하도록 Lambda를 구성할 수 있습니다.

AWS Lambda 함수에서 X-Ray 통합을 구성하려면

1. AWS Lambda 콘솔을 엽니다.
2. 왼쪽 메뉴에서 함수를 선택합니다.
3. 함수를 선택합니다.
4. 구성 탭에서 추가 모니터링 도구 카드까지 아래로 스크롤합니다. 왼쪽 탐색 창에서 모니터링 및 작업 도구를 선택하여 이 카드를 찾을 수도 있습니다.
5. 편집을 선택합니다.
6. AWS X-Ray에서 활성 추적을 활성화합니다.

해당 X-Ray SDK를 사용하는 런타임에서 Lambda는 X-Ray 대몬(daemon)도 실행합니다.

Lambda의 X-Ray SDK

- Go용 X-Ray SDK — Go 1.7 및 최신 런타임
- Java용 X-Ray SDK — 자바 8 런타임
- Node.js X-Ray SDK — Node.js 4.3 및 최신 런타임
- Python용 X-Ray SDK — Python 2.7, Python 3.6 및 최신 런타임
- .NET용 X-Ray SDK — .NET Core 2.0 및 최신 런타임

Lambda에서 X-Ray SDK를 사용하려면 새 버전을 생성할 때마다 함수 코드와 함께 번들링합니다. 다른 서비스에서 실행되는 애플리케이션을 구성할 때와 동일한 메서드를 사용하여 Lambda 함수를 구성할 수 있습니다. 주된 차이점은 수신 요청을 구성하고 샘플링 결정을 내리며 세그먼트를 생성하는 데 SDK를 사용하지 않는다는 것입니다.

Lambda 함수 구성과 웹 애플리케이션 구성 간의 기타 차이점은 Lambda가 생성하여 X-Ray로 전송하는 세그먼트를 함수 코드를 통해 수정할 수 없다는 것입니다. 하위 세그먼트를 생성하고 하위 세그먼트에 대해 주석 및 메타데이터를 기록할 수 있지만 주석 및 메타데이터를 상위 세그먼트에 추가할 수 없습니다.

자세한 내용은 AWS Lambda 개발자 안내서의 AWS X-Ray 사용을 참조하십시오.

AWS Step Functions 그리고 AWS X-Ray

AWS X-Ray 는와 통합되어 Step Functions AWS Step Functions 에 대한 요청을 추적하고 분석합니다. 상태 시스템 구성 요소를 시각화하고 성능 병목 현상을 식별하며 오류가 발생한 요청 문제를 해결할 수 있습니다. 자세한 내용은 AWS Step Functions 개발자 안내서의 AWS X-Ray 및 Step Functions를 참조하세요.

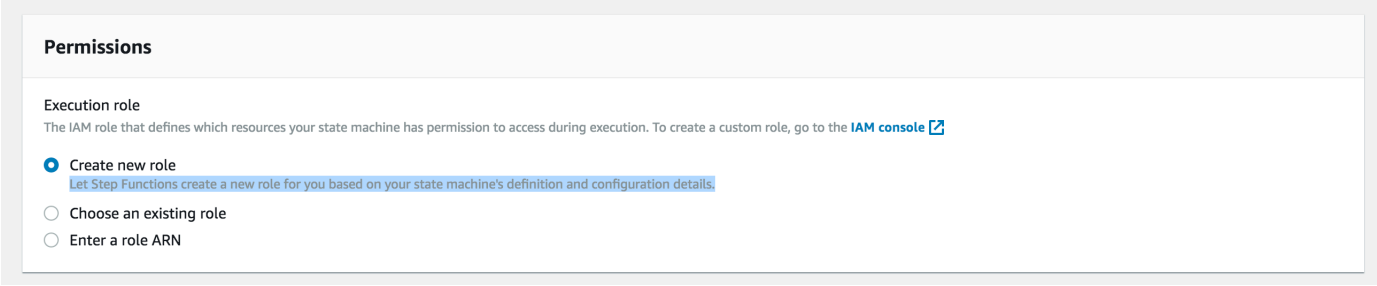
새로운 상태 시스템을 생성할 때 X-Ray 트레이싱을 활성화하기

1. https://console.aws.amazon.com/states/에서 Step Functions 콘솔을 엽니다.
2. 상태 머신 생성을 선택합니다.
3. 상태 머신 정의 페이지에서 코드 조각으로 작성하기 또는 템플릿으로 시작하기를 선택합니다. 샘플 프로젝트를 실행하기로 선택한 경우 생성 중에는 X-Ray 추적을 활성화할 수 없습니다. 대신 상태 시스템을 만든 후 X-Ray 추적을 활성화하세요.
4. Next(다음)를 선택합니다.
5. 세부 정보 지정 페이지에서 상태 시스템을 구성합니다.

6. X-Ray 추적 활성화를 선택합니다.

기존 상태 시스템에서 X-Ray 활성화하기

1. Step Functions 콘솔에서 트레이싱을 활성화하려는 상태 시스템을 선택합니다.
2. 편집을 선택합니다.
3. X-Ray 추적 활성화를 선택합니다.
4. (선택 사항) 권한 창에서 새 역할 만들기를 선택하여 상태 머신에 대한 새 역할을 자동 생성하여 X-Ray 권한을 포함할 수 있습니다.



5. 저장(Save)을 선택합니다.

> **Note**
>
> 새 상태 시스템을 생성하면 요청이 샘플링되고 Amazon API Gateway 또는 같은 업스트림 서비스에서 추적이 활성화된 경우 자동으로 추적됩니다 AWS Lambda. 예를 들어 AWS CloudFormation 템플릿을 통해 콘솔을 통해 구성되지 않은 기존 상태 시스템의 경우 X-Ray 추적을 활성화할 수 있는 충분한 권한을 부여하는 IAM 정책이 있는지 확인합니다.

용 애플리케이션 계측 AWS X-Ray

애플리케이션 계측에는 각 요청에 대한 메타데이터와 함께 애플리케이션 내의 수신 및 발신 요청 및 기타 이벤트에 대한 추적 데이터를 전송하는 작업이 포함됩니다. 특정 요구 사항에 따라 여러 가지 계측 옵션을 선택하거나 조합할 수 있습니다:

- 자동 계측 – 일반적으로 구성 변경, 자동 계측 에이전트 추가 또는 기타 메커니즘을 통해 코드 변경 없이 애플리케이션을 계측할 수 있습니다.
- 라이브러리 계측 - 애플리케이션 코드 변경을 최소화하여 AWS SDK, Apache HTTP 클라이언트 또는 SQL 클라이언트와 같은 특정 라이브러리 또는 프레임워크를 대상으로 하는 사전 구축된 계측을 추가합니다.
- 수동 계측 – 추적 정보를 전송하려는 각 위치에서 애플리케이션에 계측 코드를 추가합니다.

애플리케이션의 X-Ray 트레이싱을 위한 계측에 사용할 수 있는 여러 가지 SDK, 에이전트 및 도구가 있습니다.

주제

- AWS Distro for OpenTelemetry를 사용하여 애플리케이션 계측
- AWS X-Ray SDKs 사용하여 애플리케이션 계측
- AWS Distro for OpenTelemetry와 X-Ray SDKs 중에서 선택

AWS Distro for OpenTelemetry를 사용하여 애플리케이션 계측

AWS Distro for OpenTelemetry(ADOT)는 Cloud Native Computing Foundation(CNCF) OpenTelemetry 프로젝트를 기반으로 하는 AWS 배포입니다. OpenTelemetry는 분산된 트레이스 및 메트릭을 수집할 수 있는 단일 오픈 소스 API, 라이브러리 및 에이전트 세트를 제공합니다. 이 툴킷은 AWS에서 테스트, 최적화, 보안 및 지원하는 SDK, 자동 계측 에이전트 및 수집기를 포함한 업스트림 OpenTelemetry 구성 요소의 배포판입니다.

ADOT를 사용하면 엔지니어가 애플리케이션을 한 번 계측하고 상관관계가 있는 지표와 추적을 Amazon CloudWatch AWS X-Ray, Amazon OpenSearch Service를 포함한 여러 AWS 모니터링 솔루션으로 전송할 수 있습니다.

ADOT와 함께 X-Ray를 사용하려면 두 가지 구성 요소가 필요합니다. X-Ray와 함께 사용할 수 있는 OpenTelemetry SDK와 X-Ray와 함께 사용할 수 있는 OpenTelemetry Collector를 위한AWS 배포판입

니다. AWS X-Ray 및 기타에서 AWS Distro for OpenTelemetry를 사용하는 방법에 대한 자세한 내용은 AWS Distro for OpenTelemetry 설명서를 AWS 서비스참조하세요.

언어 지원 및 사용법에 대한 자세한 내용은 Github의AWS 관찰성을 참조하세요.

> **Note**
>
> CloudWatch 에이전트를 사용하여 Amazon EC2 인스턴스 및 온프레미스 서버에서 지표, 로그, 추적을 수집할 수 있습니다. CloudWatch 에이전트 버전 1.300025.0 이상은 OpenTelemetry 또는 X-Ray 클라이언트 SDK에서 추적을 수집하여 X-Ray에 전송할 수 있습니다. AWS Distro for OpenTelemetry(ADOT) Collector 또는 X-Ray 데몬 대신 CloudWatch 에이전트를 사용하여 추적을 수집하면 관리하는 에이전트 수를 줄이는 데 도움이 될 수 있습니다. 자세한 내용은 CloudWatch 사용 설명서의 CloudWatch 에이전트 항목을 참조하십시오.

ADOT에는 다음이 포함됩니다:

- AWS OpenTelemetry Go용 배포판
- AWS OpenTelemetry Java용 배포판
- AWS Distro for OpenTelemetry JavaScript
- AWS Distro for OpenTelemetry Python
- AWS Distro for OpenTelemetry .NET

현재 ADOT에는 Java및 Python에 대한 자동 계측 지원이 포함되어 있습니다. 또한 ADOT를 사용하면 ADOT 관리형 AWS Lambda 계층을 통해 Java, Node.js 및 Python 런타임을 사용하여 Lambda 함수 및 해당 다운스트림 요청을 자동으로 계측할 수 있습니다. https://aws-otel.github.io/docs/getting-started/lambda

Java 및 Go용 ADOT SDK는 X-Ray 통합 샘플링 규칙을 지원합니다. 다른 언어의 X-Ray 샘플링 규칙을 지원해야 하는 경우 AWS X-Ray SDK 사용을 고려해 보세요.

> **Note**
>
> 이제 W3C 추적 ID를 X-Ray로 보낼 수 있습니다. 기본적으로 OpenTelemetry로 생성된 트레이스의 트레이스 ID 형식은 W3C Trace Context 사양을 기반으로 합니다. 이는 X-Ray SDK를 사용하거나 X-Ray와 통합된 AWS 서비스에 의해 생성된 추적 IDs의 형식과 다릅니다. W3C 형식의 트레이스 ID가 X-Ray에서 허용되도록 하려면 ADOT Collector 버전 0.34.0 이상에 포함된

AWS X-Ray Exporter 버전 0.86.0 이상을 사용해야 합니다. 이전 버전의 Exporter는 트레이스 ID 타임스탬프를 검증하므로 W3C 트레이스 ID가 거부될 수 있습니다.

AWS X-Ray SDKs 사용하여 애플리케이션 계측

AWS X-Ray 에는 X-Ray로 추적을 전송하도록 애플리케이션을 계측하기 위한 언어별 SDKs 세트가 포함되어 있습니다. 각 X-Ray SDK는 다음을 제공합니다:

- 코드에 인터셉터를 추가하여 수신 HTTP 요청을 추적할 수 있습니다.
- 애플리케이션이 다른를 호출하는 데 사용하는 AWS SDK 클라이언트를 계측하는 클라이언트 핸들러 AWS 서비스
- 다른 내부 및 외부 HTTP 웹 서비스에 대한 호출을 계측하기 위한 HTTP 클라이언트

X-Ray SDKs SQL 데이터베이스, 자동 AWS SDK 클라이언트 계측 및 기타 기능에 대한 계측 호출도 지원합니다. 트레이스 데이터를 직접 X-Ray로 전송하는 대신, SDK는 UDP 트래픽을 수신 대기하는 대몬(daemon) 프로세스로 JSON 세그먼트 문서를 전송합니다. X-Ray 대몬(daemon)은 대기열에 세그먼트를 버퍼링하다가 일괄적으로 X-Ray로 업로드합니다.

다음과 같은 언어별 SDK가 제공됩니다.

- AWS X-Ray SDK for Go
- AWS X-Ray SDK for Java
- AWS X-Ray Node.js용 SDK
- AWS X-Ray SDK for Python
- AWS X-Ray SDK for .NET
- AWS X-Ray SDK for Ruby

현재 X-Ray에는 Java에 대한 자동 계측 지원이 포함되어 있습니다.

AWS Distro for OpenTelemetry와 X-Ray SDKs 중에서 선택

X-Ray에 포함된 SDK는 AWS에서 제공하는 긴밀하게 통합된 계측 솔루션의 일부입니다. OpenTelemetry용 AWS 배포판은 광범위한 산업 솔루션의 일부이며, X-Ray는 여러 추적 솔루션 중 하나에 불과합니다. 어느 접근 방식을 사용하든 X-Ray에서 엔드 투 엔드 추적을 구현할 수 있지만, 가장 유용한 접근 방식을 결정하려면 차이점을 이해하는 것이 중요합니다.

다음이 필요한 경우 AWS Distro for OpenTelemetry를 사용하여 애플리케이션을 계측하는 것이 좋습니다.

- 코드를 다시 계측할 필요 없이 여러 다른 트레이스 백엔드로 트레이스를 전송할 수 있는 기능
- OpenTelemetry 커뮤니티에서 유지 관리하는 각 언어에 대한 수많은 라이브러리 계측 도구 지원
- Java, Python 또는 Node. js 사용 시 코드를 변경할 필요 없이 원격 분석 데이터를 수집하는 데 필요한 모든 것을 패키지로 제공하는 완전 관리형 Lambda 레이어

> **Note**
>
> AWS Distro for OpenTelemetry는 Lambda 함수를 계측하기 위한 더 간단한 시작 환경을 제공합니다. 그러나 OpenTelemetry가 제공하는 유연성 때문에 Lambda 함수를 호출하려면 추가 메모리가 필요하고 콜드 스타트 지연 시간이 증가하여 추가 요금이 발생할 수 있습니다. 지연 시간이 짧도록 최적화하고 동적으로 구성 가능한 백엔드 대상과 같은 OpenTelemetry의 고급 기능이 필요하지 않은 경우 AWS X-Ray SDK를 사용하여 애플리케이션을 계측할 수 있습니다.

다음이 필요한 경우 애플리케이션 계측을 위해 X-Ray SDK를 선택하는 것을 권장합니다:

- 견고하게 통합된 단일 공급업체 솔루션
- X-Ray의 중앙 집중식 샘플링 규칙과의 통합 (Node.js, Python, Ruby 또는 .NET 사용 시 X-Ray 콘솔에서 샘플링 규칙을 구성하고 이를 여러 호스트에서 자동으로 사용하는 기능 포함)

트랜잭션 검색

트랜잭션 검색은 애플리케이션 트랜잭션 스팬을 완전하게 파악하는 데 사용할 수 있는 대화형 분석 환경입니다. 스팬은 분산 트레이스의 기본 작업 단위이며, 애플리케이션 또는 시스템의 특정 작업이나 태스크를 나타냅니다. 모든 스팬은 트랜잭션의 특정 세그먼트에 대한 세부 정보를 기록합니다. 이러한 세부 정보에는 시작 및 종료 시간, 지속 기간, 관련 메타데이터(고객 ID 및 주문 ID 같은 비즈니스 속성 포함)가 포함됩니다. 스팬은 상위-하위 계층 구조로 배열됩니다. 이 계층 구조는 다양한 구성 요소 또는 서비스에서 트랜잭션 흐름을 매핑하는 완전한 트레이스를 형성합니다.

자세한 내용은 트랜잭션 검색 섹션을 참조하세요.

OpenTelemetry Protocol(OTLP) 엔드포인트

OpenTelemetry는 원격 측정 데이터를 수집하고 라우팅하기 위한 표준화된 프로토콜과 도구를 IT 팀에 제공하는 오픈 소스 관찰성 프레임워크입니다. 이 프레임워크는 지표, 로그, 트레이스 같은 애플리케이션 원격 측정 데이터를 계측, 생성, 수집한 후 모니터링 플랫폼으로 내보내어 분석하고 인사이트를 얻기 위한 통합된 형식을 제공합니다. 팀은 OpenTelemetry를 사용하여 공급업체 종속성을 방지하여 관찰성 솔루션의 유연성을 보장할 수 있습니다.

OpenTelemetry를 사용하여 추적을 OTLP(OpenTelemetry Protocol) 엔드포인트로 직접 전송하고 CloudWatch Application Signals에서 out-of-the 애플리케이션 성능 모니터링 경험을 얻을 수 있습니다.

자세한 내용은 OpenTelemetry를 참조하세요.

Go로 작업

Go 애플리케이션을 계측하여 트레이스를 X-Ray로 보내는 방법에는 두 가지가 있습니다:

- AWS Distro for OpenTelemetry Go - AWS Distro for OpenTelemetry Collector를 통해 상관관계가 있는 지표 및 트레이스를 Amazon CloudWatch AWS X-Ray및 Amazon OpenSearch Service를 포함한 여러 AWS 모니터링 솔루션으로 전송하기 위한 오픈 소스 라이브러리 세트를 제공하는 AWS 배포입니다.
- AWS X-Ray SDK for Go - X-Ray 데몬을 통해 트레이스를 생성하고 X-Ray로 전송하기 위한 라이브러리 세트입니다.

자세한 내용은 AWS Distro for OpenTelemetry와 X-Ray SDKs 중에서 선택 단원을 참조하십시오.

AWS OpenTelemetry Go용 배포판

AWS Distro for OpenTelemetry Go를 사용하면 애플리케이션을 한 번 계측하고 상관관계가 있는 지표 및 추적을 Amazon CloudWatch AWS X-Ray및 Amazon OpenSearch Service를 포함한 여러 AWS 모니터링 솔루션으로 전송할 수 있습니다. AWS Distro for OpenTelemetry와 함께 X-Ray를 사용하려면 X-Ray에서 사용할 수 있는 OpenTelemetry SDK와 X-Ray에서 사용할 수 있는 AWS Distro for OpenTelemetry Collector라는 두 가지 구성 요소가 필요합니다.

시작하려면 OpenTelemetry Go용AWS 배포판 설명서를 참조하십시오.

AWS X-Ray 및 기타에서 AWS Distro for OpenTelemetry를 사용하는 방법에 대한 자세한 내용은 AWS Distro for OpenTelemetry 또는 AWS Distro for OpenTelemetry 설명서를 AWS 서비스참조하세요.

언어 지원 및 사용법에 대한 자세한 내용은 Github의AWS 관찰성을 참조하세요.

AWS X-Ray Go용 SDK

Go용 X-Ray SDK는 트레이스 데이터를 생성하고 X-Ray 대몬(daemon)에 보내기 위한 클래스 및 메서드를 제공하는 Go 애플리케이션용 라이브러리 집합입니다. 트레이스 데이터에는 애플리케이션에서 제공하는 수신 HTTP 요청과 애플리케이션이 AWS SDK, HTTP 클라이언트 또는 SQL 데이터베이스 커넥터를 사용하여 다운스트림 서비스에 수행하는 호출에 대한 정보가 포함됩니다. 또한 수동으로 세그먼트를 생성하고 디버그 정보를 주석 및 메타데이터에 추가할 수도 있습니다.

`go get`을 사용하여 해당 GitHub 리포지토리에서 SDK를 다운로드합니다.

```
$ go get -u github.com/aws/aws-xray-sdk-go/...
```

웹 애플리케이션의 경우 먼저 `xray.Handler` 함수를 사용하여 수신 요청을 트레이스합니다. 메시지 핸들러는 각 트레이스 요청에 대한 세그먼트를 생성하고, 응답이 전송되면 세그먼트를 완료합니다. 세그먼트가 열려 있는 동안에는 SDK 클라이언트의 메서드를 이용해 정보를 세그먼트에 추가하고 하위 세그먼트를 만들어 다운스트림 호출을 트레이스할 수 있습니다. 또한 SDK는 세그먼트가 열려 있는 동안 애플리케이션에서 발생하는 예외를 자동으로 기록합니다.

계측되는 애플리케이션 또는 서비스에 의해 호출되는 Lambda 함수의 경우, Lambda는 추적 헤더를 읽고 샘플링된 요청을 자동으로 추적합니다. 그 밖의 함수의 경우 수신 요청을 샘플링 및 추적하도록 Lambda를 구성합니다. 어느 경우든, Lambda는 세그먼트를 생성하여 X-Ray SDK에 제공합니다.

> **Note**
>
> Lambda의 X-Ray SDK는 선택 사항입니다. 이를 함수에 사용하지 않는 경우 여전히 서비스 맵에 Lambda 서비스용 노드와 각 Lambda 함수용 노트 하나가 포함됩니다. SDK를 추가하면 함수 코드를 계측하여 Lambda에 의해 기록되는 함수 세그먼트에 하위 세그먼트를 추가할 수 있습니다. 자세한 내용은 AWS Lambda 그리고 AWS X-Ray 섹션을 참조하세요.

다음으로 클라이언트를 AWS 함수에 대한 호출로 래핑합니다. 이 단계를 통해 X-Ray가 모든 클라이언트 메서드에 대한 호출을 구성할 수 있습니다. 또한 SQL 데이터베이스에 대한 호출을 구성할 수 있습니다.

SDK를 사용하기 시작한 후에, 레코더와 미들웨어를 구성하여 SDK 동작을 구성하십시오. 플러그인을 추가해 애플리케이션을 실행하는 컴퓨팅 리소스에 대한 데이터를 기록하고, 샘플링 규칙을 정의해 샘플링 동작을 구성하고, 로그 레벨을 설정해 애플리케이션 로그의 SDK에서 표시되는 정보 수준을 조절할 수 있습니다.

요청에 대한 추가 정보와 애플리케이션이 주석 및 메타데이터에서 하는 작업을 기록합니다. 주석은 필터 표현식과 함께 사용할 수 있도록 인덱싱된 단순한 키 값 쌍이기 때문에, 특정 데이터를 포함한 트레이스를 검색할 수 있습니다. 메타데이터 항목은 제한이 적으며 JSON으로 직렬화할 수 있는 모든 객체와 어레이를 기록할 수 있습니다.

> **주석 및 메타데이터**
>
> 주석 및 메타데이터는 X SDK를 사용하여 세그먼트에 추가하는 임의의 텍스트입니다. 주석은 필터 표현식에서 사용하기 위해 인덱싱됩니다. 메타데이터는 인덱싱되지 않지만 X-Ray 콘솔

또는 API를 사용하여 원시 세그먼트에서 볼 수 있습니다. X-Ray에 대한 읽기 액세스가 부여된 사용자는 누구나 이 데이터를 볼 수 있습니다.

코드에 구성된 클라이언트가 많이 있다면, 구성된 클라이언트로 만든 각 직접 호출의 하위 세그먼트를 대량으로 보관하는 요청 세그먼트 하나를 만들 수 있습니다. 사용자 지정 하위 세그먼트의 클라이언트 호출을 래핑해 하위 세그먼트를 조직하고 그룹화할 수 있습니다. 전체 함수나 특정 코드 부분에 대한 사용자 지정 하위 세그먼트를 만들고, 상위 세그먼트에 모든 것을 적는 대신 하위 세그먼트에 메타데이터와 주석을 기록할 수 있습니다.

요구 사항

Go용 X-Ray SDK를 사용하려면 Go 1.9 이상이 필요합니다.

SDK는 컴파일 및 실행 시간 시 다음 라이브러리에 의존합니다.

- AWS SDK for Go 버전 1.10.0 이상

이러한 종속성은 SDK의 `README.md` 파일에서 선언됩니다.

참조 문서

SDK를 다운로드했으면 문서를 로컬에서 빌드하고 호스트하여 웹 브라우저에서 봅니다.

참조 문서를 보려면

1. `$GOPATH/src/github.com/aws/aws-xray-sdk-go`(Linux 또는 Mac) 디렉터리 또는 `%GOPATH%\src\github.com\aws\aws-xray-sdk-go`(Windows) 폴더로 이동
2. `godoc` 명령을 실행합니다.

   ```
   $ godoc -http=:6060
   ```

3. `http://localhost:6060/pkg/github.com/aws/aws-xray-sdk-go/`에서 브라우저 열기

Go용 X-Ray SDK 구성

`Configure`을 `Config` 객체와 함께 호출하거나 기본값을 가정하여 환경 변수를 통해 Go용 X-Ray SDK에 대한 구성을 지정할 수 있습니다. 환경 변수는 모든 기본값보다 우선하는 `Config` 값보다 우선합니다.

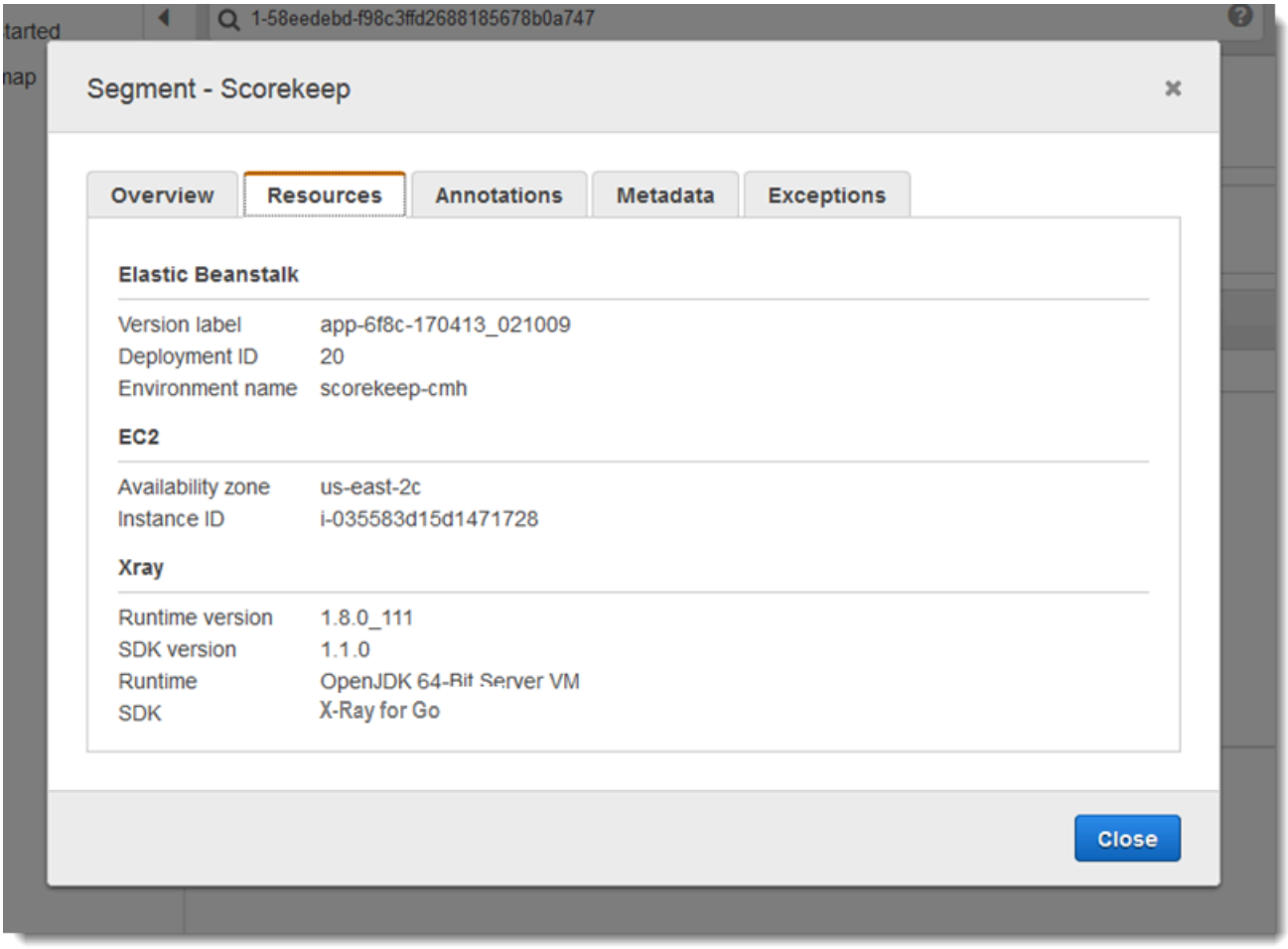
Sections

서비스 플러그인

`plugins`을 사용하여 애플리케이션을 호스팅하는 서비스에 대한 정보를 기록할 수 있습니다.

플러그인

- Amazon EC2 — `EC2Plugin`은 인스턴스 ID, 가용 영역 및 CloudWatch Logs 그룹을 추가합니다.
- Elastic Beanstalk – `ElasticBeanstalkPlugin`이 환경 이름, 버전 레이블 및 배포 ID를 추가합니다.
- Amazon ECS — `ECSPlugin`이 컨테이너 ID를 추가합니다.



플러그인을 사용하려면 다음 패키지 중 하나를 가져옵니다.

```
"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"
```

각 플러그인에는 플러그인을 로드하는 명시적 `Init()` 함수 호출이 있습니다.

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",
  })
}
```

SDK는 플러그인 설정을 사용하여 세그먼트에 `origin` 필드를 설정하기도 합니다. 이는 애플리케이션을 실행하는 AWS 리소스 유형을 나타냅니다. 여러 플러그인을 사용하는 경우 SDK는 ElasticBeanstalk> EKS> ECS> EC2 순서로 확인하여 오리진을 결정합니다.

샘플링 규칙

SDK는 X-Ray 콘솔에서 정의하는 샘플링 규칙을 사용하여 기록할 요청을 결정합니다. 기본 규칙은 매초 첫 번째 요청을 추적하고, 모든 서비스에서 추가 요청의 5%를 X-Ray로 추적 전송합니다. X-Ray 콘솔에서 추가 규칙을 생성하여 각 애플리케이션에 대해 기록되는 데이터의 양을 사용자 지정합니다.

SDK는 사용자 지정 규칙을 정의된 순서대로 적용합니다. 요청이 여러 사용자 지정 규칙과 일치하는 경우 SDK는 첫 번째 규칙만 적용합니다.

> **Note**
>
> SDK가 샘플링 규칙을 가져오기 위해 X-Ray에 연결할 수 없는 경우, 매초 첫 번째 요청과 호스트당 추가 요청의 5%에 대한 기본 로컬 규칙으로 되돌아갑니다. 호스트가 샘플링 API를 직접 호출할 수 있는 권한이 없거나, X-Ray 대몬(daemon)에 연결할 수 없을 경우 이러한 상황이 발생할 수 있고 이 대몬(daemon)은 SDK에서 수행한 API 직접 호출에 대한 TCP 프록시 역할을 합니다.

JSON 문서에서 샘플링 규칙을 불러오도록 SDK를 구성할 수도 있습니다. SDK는 X-Ray 샘플링을 사용할 수 없을 경우 로컬 규칙을 백업으로 사용하거나, 로컬 규칙을 전용으로 사용할 수 있습니다.

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
  }
}
```

이 예에서는 하나의 사용자 지정 규칙과 기본 규칙을 정의합니다. 사용자 지정 규칙은 최소 추적 요청 수 없이 5% 샘플링 비율을 `/api/move/` 아래 경로에 적용합니다. 기본 규칙은 매초 최초 요청과 추가 요청의 10%를 추적합니다.

로컬로 규칙의 정의할 때의 단점은 고정 대상이 X-Ray 서비스를 통해 관리되는 대신, 레코더의 각 인스턴스별로 독립적으로 적용된다는 것입니다. 호스트를 많이 배포할수록 고정 속도가 크게 증대하기 때문에 기록되는 데이터의 양을 제어하기가 어려워집니다.

에서는 샘플링 속도를 수정할 수 AWS Lambda없습니다. 구성된 서비스가 함수를 호출하는 경우 해당 서비스에서 샘플링한 요청을 생성한 호출이 Lambda에 의해 기록됩니다. 활성 추적이 활성화되고 트레이스 헤더가 없는 경우 Lambda에서 샘플링 결정을 내립니다.

백업 규칙을 제공하려면 `NewCentralizedStrategyWithFilePath`를 사용하여 로컬 샘플링 JSON 파일을 가리킵니다.

Example main.go – 로컬 샘플링 규칙

```
s, _ := sampling.NewCentralizedStrategyWithFilePath("sampling.json") // path to local
 sampling json
xray.Configure(xray.Config{SamplingStrategy: s})
```

로컬 규칙만 사용하려면 `NewLocalizedStrategyFromFilePath`를 사용하여 로컬 샘플링 JSON 파일을 가리킵니다.

Example main.go – 샘플링 비활성화

```
s, _ := sampling.NewLocalizedStrategyFromFilePath("sampling.json") // path to local
 sampling json
```

```
xray.Configure(xray.Config{SamplingStrategy: s})
```

로깅

> **Note**
>
> `xray.Config{}` 필드인 `LogLevel` 및 `LogFormat`은 버전 1.0.0-rc.10부터 사용 중지됩니다.

X-Ray는 로깅을 위해 다음 인터페이스를 사용합니다. 기본 로거는 `LogLevelInfo` 이상에서 `stdout`에 씁니다.

```
type Logger interface {
 Log(level LogLevel, msg fmt.Stringer)
}

const (
 LogLevelDebug LogLevel = iota + 1
 LogLevelInfo
 LogLevelWarn
 LogLevelError
)
```

Example **`io.Writer`**에 쓰기

```
xray.SetLogger(xraylog.NewDefaultLogger(os.Stderr, xraylog.LogLevelError))
```

환경 변수

환경 변수를 사용하여 Go용 X-Ray SDK를 구성할 수 있습니다. SDK는 다음 변수를 지원합니다.

- `AWS_XRAY_CONTEXT_MISSING` – 열려 있는 세그먼트가 없는 경우 구성된 구성된 코드가 데이터를 기록하려고 할 때 발생하는 예외를 방지하려면 `RUNTIME_ERROR`로 설정합니다.

  유효한 값

  - `RUNTIME_ERROR`— 런타임 예외가 발생합니다.
  - `LOG_ERROR` – 오류를 기록하고 계속합니다 (기본값).
  - `IGNORE_ERROR`— 오류를 무시하고 계속합니다.

열린 요청이 없을 때 실행되는 시작 코드 또는 새 스레드를 생성하는 코드에서 계측된 클라이언트를 사용하려고 하는 경우 누락된 세그먼트 또는 하위 세그먼트와 관련된 오류가 발생할 수 있습니다.

- `AWS_XRAY_TRACING_NAME` – SDK가 세그먼트에 사용하는 서비스 이름을 설정합니다.
- `AWS_XRAY_DAEMON_ADDRESS` – X-Ray 대몬(daemon) 리스너의 호스트와 포트를 설정합니다. 기본적으로 SDK는 `127.0.0.1:2000`에 트레이스 데이터를 보냅니다. 다른 포트에서 수신 대기하도록 대몬(daemon)을 구성한 경우 또는 다른 호스트에서 실행 중인 경우 이 변수를 사용합니다.
- `AWS_XRAY_CONTEXT_MISSING` – SDK가 누락된 컨텍스트 오류를 처리하는 방법을 결정하는 값을 설정합니다. 열린 요청이 없을 때 시작 코드 또는 새 스레드를 생성하는 코드에서 구성된 클라이언트를 사용하려고 하는 경우 누락된 세그먼트 또는 하위 세그먼트와 관련된 오류가 발생할 수 있습니다.
  - `RUNTIME_ERROR` – 기본적으로 SDK는 런타임 예외를 생성하도록 설정되어 있습니다.
  - `LOG_ERROR` – 오류를 기록하고 계속합니다.

환경 변수는 코드에 설정된 동등한 값을 재정의합니다.

configure 사용

`Configure` 메서드를 사용하여 Go용 X-Ray SDK를 구성할 수도 있습니다. `Configure`는 `Config` 객체라는 하나의 인수를 다음 선택적 필드와 함께 가져옵니다.

DaemonAddr

이 문자열은 X-Ray 대몬(daemon) 리스너의 호스트 및 포트를 지정합니다. 지정하지 않으면 X-Ray는 `AWS_XRAY_DAEMON_ADDRESS` 환경 변수의 값을 사용합니다. 해당 값이 설정되어 있지 않으면 "127.0.0.1:2000"을 사용합니다.

ServiceVersion

이 문자열은 서비스 버전을 지정합니다. 지정하지 않으면 X-Ray는 빈 문자열("")을 사용합니다.

SamplingStrategy

이 `SamplingStrategy` 객체는 트레이스되는 애플리케이션 호출을 지정합니다. 지정하지 않으면 X-Ray는 `xray/resources/DefaultSamplingRules.json`에 정의된 전략을 가져오는 `LocalizedSamplingStrategy`를 사용합니다.

StreamingStrategy

이 `StreamingStrategy` 객체는 RequiresStreaming이 true를 반환할 때 세그먼트를 스트림할지 여부를 지정합니다. 지정하지 않으면 X-Ray는 하위 세그먼트 수가 20보다 크면 샘플링된 세그먼트를 스트림하는 `DefaultStreamingStrategy`를 사용합니다.

ExceptionFormattingStrategy

이 `ExceptionFormattingStrategy` 객체는 여러 예외를 처리하는 방법을 지정합니다. 지정하지 않으면 X-Ray는 `DefaultExceptionFormattingStrategy`를 `error` 유형의 `XrayError`, 오류 메시지 및 스택 추적과 함께 사용합니다.

Go용 X-Ray SDK로 수신 HTTP 요청 계측하기

X-Ray SDK를 사용하여 애플리케이션이 Amazon EC2 AWS Elastic Beanstalk또는 Amazon ECS의 EC2 인스턴스에서 처리하는 수신 HTTP 요청을 추적할 수 있습니다.

`xray.Handler`를 사용하여 수신 HTTP 요청을 구성합니다. Go용 X-Ray SDK는 `xray.Handler` 클래스에서 표준 Go 라이브러리 `http.Handler` 인터페이스를 구현하여 웹 요청을 가로챕니다. `xray.Handler` 클래스는 요청의 컨텍스트를 사용하고, 수신 헤더를 구문 분석하고, 필요한 경우 응답 헤더를 추가하여 `xray.Capture`로 제공된 `http.Handler`를 래핑하고 HTTP별 트레이스 필드를 설정합니다.

이 클래스를 사용하여 HTTP 요청 및 응답을 처리하면 Go용 X-Ray SDK는 샘플링된 각 요청에 대한 세그먼트를 생성합니다. 이 세그먼트에는 HTTP 요청의 시간, 메서드 및 배치가 포함됩니다. 추가로 구성하면 이 세그먼트의 하위 세그먼트가 생성됩니다.

> **Note**
>
> AWS Lambda 함수의 경우 Lambda는 샘플링된 각 요청에 대해 세그먼트를 생성합니다. 자세한 내용은 AWS Lambda 그리고 AWS X-Ray 섹션을 참조하세요.

다음 예제에서는 포트 8000의 요청을 가로채고 "Hello!"를 응답으로 반환합니다. myApp 세그먼트를 생성하고 애플리케이션을 통해 호출을 구성합니다.

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)
```

```
}
```

각 세그먼트에는 서비스 맵 안에서 애플리케이션을 식별하는 이름이 있습니다. 이 세그먼트의 이름이 정적으로 지정되도록 하거나, 수신 요청의 호스트 헤더를 기반으로 SDK가 동적으로 이름을 지정하도록 구성할 수 있습니다. 동적 이름 지정을 사용하면 요청의 도메인 이름에 따라 그룹을 추적하고 이름이 예상 패턴과 일치하지 않을 경우(예: 호스트 헤더가 위조된 경우) 기본 이름을 적용할 수 있습니다.

> **전달된 요청**
>
> 로드 밸런서 또는 기타 중개자가 애플리케이션으로 요청을 전달하는 경우, X-Ray는 IP 패킷 내 소스 IP가 아니라 요청의 `X-Forwarded-For` 헤더로부터 클라이언트 IP를 가져옵니다. 전달된 요청에 대해 기록된 클라어언트 IP는 위조될 수 있으므로 신뢰하면 안 됩니다.

요청이 전달되면 SDK는 세그먼트에 추가 필드를 설정하여 이를 나타냅니다. 세그먼트에 `x_forwarded_for`로 설정된 `true` 필드가 포함된 경우 클라이언트 IP는 HTTP 요청의 `X-Forwarded-For` 헤더로부터 가져옵니다.

핸들러는 다음 정보를 포함하는 `http` 블록으로 각 수신 요청에 대한 세그먼트를 생성합니다.

- HTTP 메서드 – GET, POST, PUT, DELETE 등.
- 클라이언트 주소 – 요청을 전송한 클라이언트의 IP 주소.
- 응답 코드 – 완료된 요청의 HTTP 응답 코드.
- 시간 – 시작 시간(요청 수신) 및 종료 시간(응답 전송).
- 유저 에이전트 — 요청에서 가져온 `user-agent`입니다.
- 콘텐츠 길이 — 응답의 `content-length`입니다.

세그먼트 이름 지정 전략 구성

AWS X-Ray 는 서비스 이름을 사용하여 애플리케이션을 식별하고 애플리케이션이 사용하는 다른 애플리케이션, 데이터베이스, 외부 APIs 및 AWS 리소스와 구분합니다. X-Ray SDK는 수신 요청에 대한 세그먼트를 생성할 때 해당 세그먼트의 이름 필드에 애플리케이션의 서비스 이름을 기록합니다.

X-Ray SDK는 HTTP 요청 헤더의 호스트 이름 뒤에 세그먼트를 지정할 수 있습니다. 그러나 이 헤더가 위조되면 서비스 맵에 예기치 않은 노드가 발생할 수 있습니다. 위조된 호스트 헤더가 포함된 요청으로 인해 SDK가 잘못된 세그먼트 이름을 지정하는 현상을 방지하려면 들어오는 요청의 기본 이름을 지정해야 합니다.

애플리케이션이 여러 도메인의 요청을 처리하는 경우, 동적 이름 지정 전략을 사용하여 이를 세그먼트 이름에 반영하도록 SDK를 구성할 수 있습니다. 동적 이름 지정 전략을 사용하면 SDK가 예상 패턴과 일치하는 요청에 호스트 이름을 사용하고, 그렇지 않은 요청에 기본 이름을 적용할 수 있습니다.

예를 들어, 하나의 애플리케이션이 세 개의 하위 도메인 (`www.example.com`, `api.example.com`, `static.example.com`) 에 요청을 전송할 수 있습니다. `*.example.com` 패턴으로 동적 이름 지정 전략을 사용하여 각 하위도메인의 세그먼트를 서로 다른 이름으로 표시하면 서비스 맵에 서비스 노드가 세 개 생깁니다. 이 패턴에 맞지 않는 호스트 이름의 요청이 애플리케이션에 수신되면, 사용자가 지정한 대체 이름의 네 번째 노드가 서비스 맵에 표시됩니다.

모든 요청 세그먼트에 같은 이름을 사용하려면 이전 섹션과 같이 핸들러를 생성할 때 애플리케이션 이름을 지정합니다.

> **Note**
>
> 코드에 정의한 기본 서비스 이름을 `AWS_XRAY_TRACING_NAME` 환경 변수를 사용하여 재정의할 수 있습니다.

동적 이름 지정 전략은 호스트 이름이 일치해야 하는 패턴 및 HTTP 요청의 호스트 이름이 패턴과 일치하지 않는 경우 사용할 기본 이름을 정의합니다. 세그먼트 이름을 동적으로 지정하려면 `NewDynamicSegmentNamer`를 사용하여 일치하는 패턴 및 기본 이름을 구성합니다.

Example main.go

요청의 호스트 이름이 `*.example.com` 패턴과 일치하는 경우 호스트 이름을 사용합니다. 그렇지 않은 경우 `MyApp`을 사용합니다.

```
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)
}
```

Go용 X-Ray AWS SDK를 사용하여 SDK 호출 추적

애플리케이션이를 호출 AWS 서비스 하여 데이터를 저장하거나, 대기열에 쓰거나, 알림을 보내면 Go용 X-Ray SDK는 하위 세그먼트에서 다운스트림으로 호출을 추적합니다. 해당 서비스(예: Amazon S3 버킷 또는 Amazon SQS 대기열) 내에서 액세스하는 추적 AWS 서비스 및 리소스는 X-Ray 콘솔의 추적 맵에 다운스트림 노드로 표시됩니다.

AWS SDK 클라이언트를 트레이스하려면 다음 예와 같이 `xray.AWS()` 호출로 클라이언트 객체를 래핑합니다.

Example main.go

```
var dynamo *dynamodb.DynamoDB
func main() {
  dynamo = dynamodb.New(session.Must(session.NewSession()))
  xray.AWS(dynamo.Client)
}
```

그런 다음 AWS SDK 클라이언트를 사용할 때 직접 호출 메서드의 `withContext` 버전을 사용하고 핸들러에 전달된 `http.Request` 객체에서 `context`를 전달합니다.

Example main.go – AWS SDK 호출

```
func listTablesWithContext(ctx context.Context) {
  output := dynamo.ListTablesWithContext(ctx, &dynamodb.ListTablesInput{})
  doSomething(output)
}
```

모든 서비스의 경우, X-Ray 콘솔에서 호출된 API의 이름을 볼 수 있습니다. 서비스 하위 집합에 대해서는 X-Ray SDK가 세그먼트에 정보를 추가하여 서비스 맵에서 추가 세분화를 제공합니다.

예를 들어 계측된 DynamoDB 클라이언트에서 직접 호출을 생성하는 경우 SDK가 특정 테이블을 대상으로 한 직접 호출에 대해 테이블 이름을 세그먼트에 추가합니다. 콘솔에서, 각 테이블이 개별 노드로 서비스 맵에 표시되고, 특정 테이블을 대상으로 하지 않은 직접 호출에 대해 일반 DynamoDB 노드가 표시됩니다.

Example 항목을 저장하기 위한 DynamoDB 직접 호출에 대한 하위 세그먼트

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

명명된 리소스에 액세스할 때 다음 서비스를 호출할 경우 서비스 맵에 추가 노드가 생성됩니다. 특정 리소스를 대상으로 하지 않는 경우 서비스를 직접 호출하면 해당 서비스에 대한 일반 노드가 생성됩니다.

- Amazon DynamoDB – 테이블 이름
- Amazon Simple Storage Service –버킷 및 키 이름
- Amazon Simple Queue Service – 대기열 이름

Go용 X-Ray SDK를 사용하여 다운스트림 HTTP 웹 서비스에 대한 호출 추적하기

애플리케이션이 마이크로서비스 또는 퍼블릭 HTTP API를 호출하면 다음 예제와 같이 `xray.Client`를 사용하여 이러한 호출을 Go 애플리케이션의 하위 세그먼트로 구성할 수 있습니다. 이 예제에서 http-client는 HTTP 클라이언트입니다.

클라이언트는 `xray.RoundTripper`로 래핑된 라운드트리퍼와 함께 기본값이 `http.DefaultClient`로 설정되는 제공된 http 클라이언트의 단순 복사본을 생성합니다.

Example

<caption>main.go – HTTP 클라이언트</caption>

```
myClient := xray.Client(http-client)
```

<caption>main.go – ctxhttp 라이브러리를 사용하여 다운스트림 HTTP 직접 호출을 추적합니다.</caption>

다음 예제는 `xray.Client`를 사용하여 ctxhttp 라이브러리로 발신 HTTP 직접 호출을 계측합니다. 업스트림 호출에서 `ctx`를 전달할 수 있습니다. 이는 기존 세그먼트 컨텍스트가 사용되도록 보장합니다. 예를 들어, X-Ray에서는 Lambda 함수 내에 새 세그먼트를 생성할 수 없으므로 기존 Lambda 세그먼트 컨텍스트를 사용해야 합니다.

```
resp, err := ctxhttp.Get(ctx, xray.Client(nil), url)
```

Go용 X-Ray SDK로 SQL 쿼리 추적하기

PostgreSQL 또는 MySQL에 대한 SQL 호출을 트레이스하려면 다음 예와 같이 `sql.Open`에 대한 `xray.SQLContext` 호출을 교체합니다. 가능하면 구성 문자열 대신 URL을 사용합니다.

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
}
```

Go용 X-Ray SDK를 사용하여 사용자 지정 하위 세그먼트 생성하기

하위 세그먼트는 추적의 세그먼트를 확장하여 요청을 처리하기 위해 완료된 작업에 대한 세부 정보를 표시합니다. 계측되는 클라이언트에서 직접 호출할 때마다, X-Ray SDK는 하위 세그먼트 안에 생성된 정보를 기록합니다. 추가 하위 세그먼트를 생성하여 다른 하위 세그먼트를 그룹화하거나, 코드 섹션의 성능을 평가하거나, 주석 및 메타데이터를 기록할 수 있습니다.

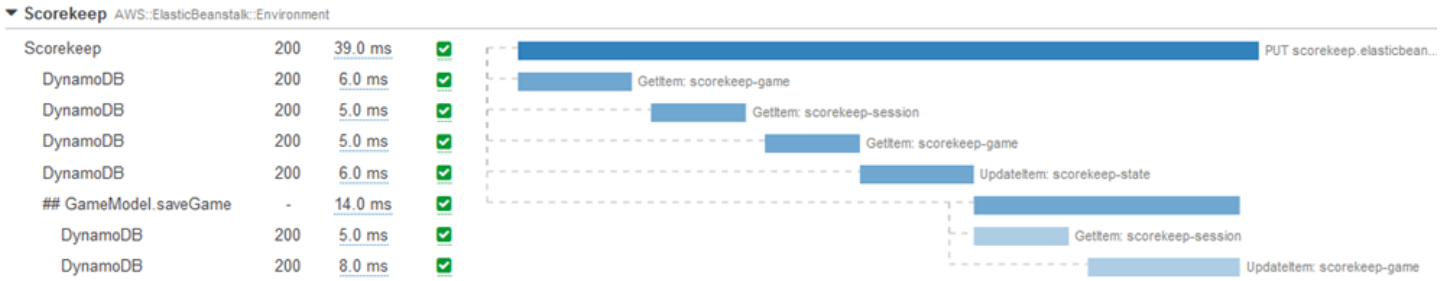
`Capture` 메서드를 사용하여 함수 주변에 하위 세그먼트를 생성합니다.

Example main.go – 사용자 지정 하위 세그먼트

```
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)
  })
```

다음 스크린샷은 Scorekeep 애플리케이션에 대한 추적에 saveGame 하위 세그먼트가 나타나는 방법의 예를 보여줍니다.



Go용 X-Ray SDK로 세그먼트에 주석 및 메타데이터 추가하기

주석 및 메타데이터와 함께 요청, 환경 또는 애플리케이션에 대한 추가 정보를 기록할 수 있습니다. X-Ray SDK에서 생성하는 세그먼트 또는 사용자가 생성하는 사용자 지정 하위 세그먼트에 주석 및 메타데이터를 추가할 수 있습니다.

주석은 문자열, 숫자 또는 부울 값과 결합한 키-값 페어입니다. 주석은 필터 표현식에서 사용하기 위해 인덱싱됩니다. 주석은 콘솔의 트레이스를 그룹화할 때 사용할 데이터를 기록하거나 `GetTraceSummaries` API를 직접 호출할 때 사용하세요.

메타데이터는 객체 및 목록을 포함한 모든 유형의 값을 가질 수 있는 키-값 페어지만, 필터 표현식에 사용할 수 있도록 인덱싱되지는 않습니다. 트레이스에 저장하고 싶지만 검색에는 사용하지 않을 추가 데이터는 메타데이터를 사용하여 기록하십시오.

세그먼트에는 주석과 메타데이터 외에 사용자 ID 문자열도 기록할 수 있습니다. 사용자 ID는 세그먼트의 별도 필드에 기록되면 검색용으로 인덱스되지 않습니다.

Sections

- Go용 X-Ray SDK로 주석 기록하기
- Go용 X-Ray SDK로 메타데이터 기록하기
- Go용 X-Ray SDK로 사용자 ID 기록하기

Go용 X-Ray SDK로 주석 기록하기

주석을 사용하여 검색용으로 인덱싱할 세그먼트에 정보를 레코딩할 수 있습니다.

주석 요구 사항

- 키 - X-Ray 주석의 키는 최대 500자의 영숫자를 포함할 수 있습니다. 점이나 마침표(.) 이외의 공백이나 기호를 사용할 수 없습니다.
- 값 - X-Ray 주석의 값은 최대 1,000자의 유니코드 문자를 포함할 수 있습니다.
- 주석 수 - 트레이스당 최대 50개의 주석을 사용할 수 있습니다.

주석을 레코딩하려면 세그먼트와 연결할 메타데이터를 포함하는 문자열과 함께 `AddAnnotation`을 호출합니다.

```
xray.AddAnnotation(key string, value interface{})
```

SDK는 세그먼트 문서의 `annotations` 객체에 주석을 키-값 페어로 기록합니다. 동일한 키로 `AddAnnotation`을 두 번 호출하면 동일한 세그먼트에 레코딩했던 값을 덮어씁니다.

특정 값을 포함한 주석이 있는 트레이스를 찾으려면 `annotation[key]` 키워드를 필터 표현식에 사용하십시오.

Go용 X-Ray SDK로 메타데이터 기록하기

메타데이터를 사용하여 검색용으로 인덱싱할 필요가 없는 세그먼트에 정보를 레코딩합니다.

메타데이터를 레코딩하려면 세그먼트와 연결할 메타데이터를 포함하는 문자열과 함께 `AddMetadata`를 호출합니다.

```
xray.AddMetadata(key string, value interface{})
```

Go용 X-Ray SDK로 사용자 ID 기록하기

사용자 ID를 요청 세그먼트에 기록하여 요청을 보낸 사용자를 식별합니다.

사용자 ID 기록 방법

1. `AWSXRay`에서 현재 세그먼트에 대한 참조를 가져옵니다.

```
import (
  "context"
  "github.com/aws/aws-xray-sdk-go/xray"
)
```

```
mySegment := xray.GetSegment(context)
```

2. 요청을 보낸 사용자의 문자열 ID로 `setUser`를 직접 호출합니다.

```
mySegment.User = "U12345"
```

사용자 ID의 트레이스를 찾으려면, `user` 키워드를 필터 표현식에 적용하십시오.

Java 작업

Java 애플리케이션을 계측하여 트레이스를 X-Ray로 전송하는 방법에는 두 가지가 있습니다.

- AWS Distro for OpenTelemetry Java - AWS Distro for OpenTelemetry Collector를 통해 상관관계가 있는 지표 및 트레이스를 Amazon CloudWatch AWS X-Ray및 Amazon OpenSearch Service를 포함한 여러 AWS 모니터링 솔루션으로 전송하기 위한 오픈 소스 라이브러리 세트를 제공하는 AWS 배포입니다.
- AWS X-Ray Java용 SDK - X-Ray 데몬을 통해 트레이스를 생성하고 X-Ray로 전송하기 위한 라이브러리 세트입니다.

자세한 내용은 AWS Distro for OpenTelemetry와 X-Ray SDKs 중에서 선택 단원을 참조하십시오.

AWS OpenTelemetry Java용 배포판

AWS Distro for OpenTelemetry(ADOT) Java를 사용하면 애플리케이션을 한 번 계측하고 상관관계가 있는 지표 및 추적을 Amazon CloudWatch AWS X-Ray, Amazon OpenSearch Service를 포함한 여러 AWS 모니터링 솔루션으로 전송할 수 있습니다. ADOT와 함께 X-Ray를 사용하려면 두 가지 구성 요소가 필요합니다. X-Ray와 함께 사용할 수 있는 OpenTelemetry SDK와 X-Ray와 함께 사용할 수 있는 OpenTelemetry Collector를 위한AWS 배포판입니다. ADOT Java에는 자동 계측 지원이 포함되어 애플리케이션에서 코드 변경 없이 트레이스를 전송할 수 있습니다.

시작하려면 Java용 OpenTelemetry AWS 배포판 설명서를 참조하십시오.

AWS X-Ray 및 기타에서 AWS Distro for OpenTelemetry를 사용하는 방법에 대한 자세한 내용은 AWS Distro for OpenTelemetry 또는 AWS Distro for OpenTelemetry 설명서를 AWS 서비스참조하세요.

언어 지원 및 사용법에 대한 자세한 내용은 Github의AWS 관찰성을 참조하세요.

AWS X-Ray Java용 SDK

Java용 X-Ray SDK는 트레이스 데이터를 생성하고 X-Ray 대몬(daemon)에 보내기 위한 클래스 및 메서드를 제공하는 Java 웹 애플리케이션용 라이브러리 집합입니다. 트레이스 데이터에는 애플리케이션에서 제공하는 수신 HTTP 요청과 애플리케이션이 AWS SDK, HTTP 클라이언트 또는 SQL 데이터베이스 커넥터를 사용하여 다운스트림 서비스에 수행하는 호출에 대한 정보가 포함됩니다. 또한 수동으로 세그먼트를 생성하고 디버그 정보를 주석 및 메타데이터에 추가할 수도 있습니다.

Java용 X-Ray SDK는 오픈 소스 프로젝트입니다. 프로젝트를 따르고 GitHub(github.com/aws/aws-xray-sdk-java)에서 문제 및 풀 요청을 제출할 수 있습니다.

AWSXRayServletFilter를 서블릿 필터로 츄가하여 수신 요청 트레이스를 시작합니다. 서블릿 필터가 세그먼트를 생성합니다. 세그먼트가 열려 있는 동안에는 SDK 클라이언트의 메서드를 이용해 정보를 세그먼트에 추가하고 하위 세그먼트를 만들어 다운스트림 호출을 트레이스할 수 있습니다. 또한 SDK는 세그먼트가 열려 있는 동안 애플리케이션에서 발생하는 예외를 자동으로 기록합니다.

릴리스 1.3부터는 Spring에서 관점 지향 프로그래밍(AOP)을 사용하여 애플리케이션을 계측할 수 있습니다. 즉, 애플리케이션을 실행하는 동안 애플리케이션의 런타임에 코드를 추가하지 AWS않고도 애플리케이션을 계측할 수 있습니다.

그런 다음 Java용 X-Ray SDK를 사용하여 빌드 구성에 SDK Instrumentor 하위 모듈을 포함하여 AWS SDK for Java 클라이언트를 계측합니다. ??? 계측된 클라이언트를 사용하여 다운스트림 AWS 서비스 또는 리소스를 호출할 때마다 SDK는 하위 세그먼트에 호출에 대한 정보를 기록합니다. 서비스 내에서 액세스하는 AWS 서비스 리소스는 트레이스 맵에 다운스트림 노드로 표시되므로 개별 연결에서 오류 및 제한 문제를 식별하는 데 도움이 됩니다.

모든 다운스트림 호출을 계측하지 않으려면 계측기 하위 모듈을 생략하고 계측할 클라이언트를 선택할 AWS 서비스수 있습니다. AWS SDK 서비스 클라이언트에 를 추가하여 TracingHandler 개별 클라이언트를 계측합니다.

다른 Java용 X-Ray SDK 하위 모듈은 HTTP web API 및 SQL 데이터베이스에 대한 다운스트림 호출을 계측합니다. Apache HTTP 하위 모듈의 Java용 X-Ray SDK 버전 HTTPClient와 HTTPClientBuilder를 사용하여 Apache HTTP 클라이언트를 계측할 수 있습니다. SQL 쿼리를 구성하려면 SDK의 인터셉터를 데이터 원본에 추가합니다.

SDK를 사용하기 시작한 후에, 레코더와 servlet 필터를 구성하여 SDK 동작을 구성하십시오. 플러그인을 추가해 애플리케이션을 실행하는 컴퓨팅 리소스에 대한 데이터를 기록하고, 샘플링 규칙을 정의해 샘플링 동작을 구성하고, 로그 레벨을 설정해 애플리케이션 로그의 SDK에서 표시되는 정보 수준을 조절할 수 있습니다.

요청에 대한 추가 정보와 애플리케이션이 주석 및 메타데이터에서 하는 작업을 기록합니다. 주석은 필터 표현식과 함께 사용할 수 있도록 인덱싱된 단순한 키 값 쌍이기 때문에, 특정 데이터를 포함한 트레이스를 검색할 수 있습니다. 메타데이터 항목은 제한이 적으며 JSON으로 직렬화할 수 있는 모든 객체와 어레이를 기록할 수 있습니다.

> **주석 및 메타데이터**
>
> 주석 및 메타데이터는 X SDK를 사용하여 세그먼트에 추가하는 임의의 텍스트입니다. 주석은 필터 표현식에서 사용하기 위해 인덱싱됩니다. 메타데이터는 인덱싱되지 않지만 X-Ray 콘솔 또는 API를 사용하여 원시 세그먼트에서 볼 수 있습니다. X-Ray에 대한 읽기 액세스가 부여된 사용자는 누구나 이 데이터를 볼 수 있습니다.

코드에 계측된 클라이언트가 많다면, 계측된 클라이언트로 만든 각 요청의 하위 세그먼트를 대량으로 보관하는 요청 세그먼트 하나를 만들 수 있습니다. 사용자 지정 하위 세그먼트의 클라이언트 호출을 래핑해 하위 세그먼트를 조직하고 그룹화할 수 있습니다. 전체 함수나 특정 코드 부분에 대한 사용자 지정 하위 세그먼트를 만들고, 상위 세그먼트에 모든 것을 적는 대신 하위 세그먼트에 메타데이터와 주석을 기록할 수 있습니다.

하위 모듈

Maven에서 Java용 X-Ray SDK를 다운로드할 수 있습니다. Java용 X-Ray SDK는 사용 사례별로 하위 모듈로 나뉘며, 버전 관리용 BOM을 포함합니다:

- `aws-xray-recorder-sdk-core` (필수) – 세그먼트 생성 및 세그먼트 전송을 위한 기본 기능. 수신 요청 구성을 위한 `AWSXRayServletFilter`를 포함합니다.
- `aws-xray-recorder-sdk-aws-sdk` - 추적 AWS SDK for Java 클라이언트를 요청 핸들러로 추가하여 클라이언트에서 AWS 서비스 수행된 호출을 계측합니다.
- `aws-xray-recorder-sdk-aws-sdk-v2` - 추적 클라이언트를 요청 인터셉터로 추가하여 AWS SDK for Java 2.2 이상 클라이언트에서 AWS 서비스 수행된 호출을 계측합니다.
- `aws-xray-recorder-sdk-aws-sdk-instrumentor` - `aws-xray-recorder-sdk-aws-sdk`를 사용하여 모든 AWS SDK for Java 클라이언트를 자동으로 계측합니다.
- `aws-xray-recorder-sdk-aws-sdk-v2-instrumentor` - `aws-xray-recorder-sdk-aws-sdk-v2`를 사용하면 모든 AWS SDK for Java 2.2 이상 클라이언트를 자동으로 계측합니다.
- `aws-xray-recorder-sdk-apache-http` – Apache HTTP 클라이언트에서 생성된 발신 HTTP 호출을 구성합니다.
- `aws-xray-recorder-sdk-spring` – Spring AOP 프레임워크 애플리케이션의 인터셉터를 제공합니다.
- `aws-xray-recorder-sdk-sql-postgres` – JDBC에서 생성된 PostgreSQL 데이터베이스에 대한 발신 호출을 구성합니다.

- aws-xray-recorder-sdk-sql-mysql – JDBC에서 생성된 MySQL 데이터베이스에 대한 발신 호출을 구성합니다.
- aws-xray-recorder-sdk-bom – 모든 하위 모듈에 사용할 버전을 지정하는 데 사용할 수 있는 BOM을 제공합니다.
- aws-xray-recorder-sdk-metrics – 수집된 X-Ray 세그먼트에서 샘플링되지 않은 Amazon CloudWatch 지표를 게시합니다.

Maven 또는 Gradle을 사용하여 애플리케이션을 빌드하는 경우, 빌드 구성에 Java용 X-Ray SDK를 추가합니다.

SDK의 클래스 및 메서드에 대한 참조 문서는 Java용AWS X-Ray SDK API Reference를 참조하십시오.

요구 사항

Java용 X-Ray SDK에는 Java 8 이상, Servlet API 3, AWS SDK 및 Jackson이 필요합니다.

SDK는 컴파일 및 실행 시간 시 다음 라이브러리에 의존합니다.

- AWS SDK for Java 버전 1.11.398 이상
- Servlet API 3.1.0

이러한 종속성은 SDK의 `pom.xml` 파일에서 선언되며 Maven 또는 Gradle을 사용하여 빌드하는 경우 자동으로 포함됩니다.

Java용 X-Ray SDK에 포함된 라이브러리를 사용하는 경우 포함된 버전을 사용해야 합니다. 예를 들어 실행 시간 시 이미 Jackson에 의존하고 해당 종속성을 위해 배포에 JAR 파일을 포함시킨 경우에는 SDK JAR이 자체 버전의 Jackson 라이브러리를 포함하므로 이들 JAR 파일을 제거해야 합니다.

종속성 관리

Java용 X-Ray SDK는 Maven에서 사용할 수 있습니다.

- 그룹 – `com.amazonaws`
- 아티팩트 – `aws-xray-recorder-sdk-bom`
- 버전 – `2.11.0`

Maven을 사용하여 애플리케이션을 빌드하는 경우 pom.xml 파일에서 SDK를 종속성으로 추가합니다.

Example pom.xml - 종속성

```
<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>
```

Gradle의 경우, `build.gradle` 파일에서 SDK를 컴파일 시점 종속성으로 추가합니다.

Example build.gradle - 종속성

```
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')
    }
}
```

Elastic Beanstalk를 사용하여 애플리케이션을 배포하는 경우, 모든 종속 요소를 포함하는 대규모 아카이브를 빌드하고 업로드하는 대신 배포할 때마다 Maven 또는 Gradle을 사용하여 온-인스턴스를 빌드할 수 있습니다. Gradle 사용 예시를 보려면 샘플 애플리케이션을 참조하십시오.

Java용AWS X-Ray 자동 계측 에이전트

Java용 AWS X-Ray 자동 계측 에이전트는 최소한의 개발 노력으로 Java 웹 애플리케이션을 계측하는 추적 솔루션입니다. 이 에이전트는 서블릿 기반 애플리케이션과 지원되는 프레임워크 및 라이브러리로 이루어진 에이전트의 모든 다운스트림 요청에 대한 추적을 지원합니다. 여기에는 다운스트림 Apache HTTP 요청, AWS SDK 요청 및 JDBC 드라이버를 사용하여 만든 SQL 쿼리가 포함됩니다. 에이전트는 모든 활성 세그먼트와 하위 세그먼트를 포함한 X-Ray 컨텍스트를 스레드 전체에 전파합니다. X-Ray SDK의 모든 구성과 다양한 기능을 Java 에이전트에서 계속 사용할 수 있습니다. 에이전트가 최소한의 노력으로 작업할 수 있도록 적절한 기본값이 선택되었습니다.

X-Ray 에이전트 솔루션은 서블릿 기반의 요청-응답 Java 웹 애플리케이션 서버에 가장 적합합니다. 애플리케이션이 비동기 프레임워크를 사용하거나 요청-응답 서비스로 잘 모델링되지 않은 경우, SDK를 사용한 수동 계측을 대신 고려할 수 있습니다.

X-Ray 에이전트는 분산 시스템 이해 툴킷(DiSCo)을 사용하여 구축됩니다. DiSCo는 분산 시스템에서 사용할 수 있는 Java 에이전트를 구축하기 위한 오픈 소스 프레임워크입니다. X-Ray 에이전트를 사용하기 위해 DisCo를 이해할 필요는 없지만 GitHub의 홈페이지에서 프로젝트에 대해 자세히 알아볼 수 있습니다. 또한 X-Ray 에이전트는 완전히 오픈 소스입니다. 소스 코드를 보거나, 기여하거나, 에이전트에 대한 문제를 제기하려면 GitHub의 해당 리포지토리를 방문하세요.

샘플 애플리케이션

eb-java-scorekeep 샘플 애플리케이션은 X-Ray 에이전트로 계측할 수 있도록 조정되었습니다. 이러한 기능은 에이전트가 수행하므로 이 브랜치에는 서블릿 필터나 레코더 구성이 포함되어 있지 않습니다. 애플리케이션을 로컬에서 실행하거나 AWS 리소스를 사용하여 실행하려면 샘플 애플리케이션의 readme 파일에 있는 단계를 따르세요. 샘플 앱을 사용하여 X-Ray 트레이스를 생성하는 방법은 샘플 앱의 튜토리얼에 나와 있습니다.

시작

자체 애플리케이션에서 X-Ray 자동 계측 Java 에이전트를 시작하려면 다음 절차를 따르십시오.

1. 사용 중인 환경에서 X-Ray 대몬(daemon)을 실행합니다. 자세한 내용은 AWS X-Ray 데몬 단원을 참조하십시오.
2. 에이전트의 최신 배포판을 다운로드하십시오. 아카이브의 압축을 풀고 파일 시스템에 해당 위치를 기록해 둡니다. 내용은 다음과 같아야 합니다.

```
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. 애플리케이션의 JVM 인수가 다음을 포함하도록 수정하여 에이전트를 활성화합니다. 해당하는 경우 `-jar` 인수 앞에 `-javaagent` 인수를 배치해야 합니다. JVM 인수를 수정하는 프로세스는 Java 서버를 시작하는 데 사용하는 도구와 프레임워크에 따라 다릅니다. 구체적인 지침은 사용 중인 서버 프레임워크의 설명서를 참조하세요.

```
-javaagent:/<path-to-disco>/disco-java-agent.jar=pluginPath=/<path-to-disco>/disco-
plugins
```

4. 응용 프로그램 이름이 X-Ray 콘솔에 표시되는 방식을 지정하려면 `AWS_XRAY_TRACING_NAME` 환경 변수 또는 `com.amazonaws.xray.strategy.tracingName` 시스템 속성을 설정하십시오. 이름을 지정하지 않으면 기본 이름이 사용됩니다.
5. 서버 또는 컨테이너를 다시 시작합니다. 이제 수신 요청과 해당 다운스트림 호출이 추적됩니다. 예상 결과가 표시되지 않는 경우 the section called “문제 해결”을 참조하십시오.

구성

X-Ray 에이전트는 사용자가 제공한 외부 JSON 파일로 구성됩니다. 기본적으로 이 파일은 이름이 지정된 사용자 클래스 경로 (예: `resources` 디렉터리) `xray-agent.json`의 루트에 있습니다. `com.amazonaws.xray.configFile` 시스템 속성을 구성 파일의 절대 파일 시스템 경로로 설정하여 구성 파일의 사용자 지정 위치를 구성할 수 있습니다.

다음은 구성 파일의 예시입니다.

```
{
    "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
}
```

구성 사양

다음 표에서는 각 속성의 유효한 값에 대해 설명합니다. 속성 이름은 대소문자를 구분하지만 키는 대소문자를 구분하지 않습니다. 환경 변수 및 시스템 속성으로 재정의할 수 있는 속성의 경우 우선 순위는 항상 환경 변수, 시스템 속성, 구성 파일 순입니다. 재정의할 수 있는 속성에 대한 자세한 내용은 환경 변수 단원을 참조하세요. 모든 필드는 선택 사항입니다.

속성 이름	유형	유효값	설명	환경 변수	시스템 속성	Default
serviceName	String	모든 문자열	X-Ray 콘솔에 표시되는 계측된 서비스의 이름	AWS_XRAY_TRACING_NAME	com.amazonaws.xray.Strategy.tracingName	XRayInstrumentedService
contextMissingStrategy	String	LOG_ERROR, IGNORE_ERROR	에이전트가 X-Ray 세그먼트 컨텍스트를 사용하려고 시도했지만 아무것도 없을 때 에이전트가 취하는 작업입니다.	AWS_XRAY_CONTEXT_MISSING	com.amazonaws.xray.Strategy.ContextMissingStrategy	LOG_ERROR
daemonAddress	String	포맷된 IP 주소 및 포트 또는 TCP 및 UDP 주소 목록	에이전트가 X-Ray 대몬(daemon)과 통신하는 데 사용하는 주소입니다.	AWS_XRAY_DAEMON_ADDRESS	com.amazonaws.xray.emitter.daemonAddress	127.0.0.1:2000
tracingEnabled	불	True, False	엑스레이 에이전트에 의한 계측을 활성화합니다.	AWS_XRAY_TRACING_ENABLED	com.amazonaws.xray.tracingEnabled	TRUE
samplingStrategy	String	CENTRAL, LOCAL,	에이전트가 사용하는	N/A	N/A	CENTRAL

속성 이름	유형	유효값	설명	환경 변수	시스템 속성	Default
		NONE, ALL	샘플링 전략. ALL은 모든 요청을 캡처하고, NONE은 요청을 캡처하지 않습니다. 샘플링 규칙을 참조하세요.			
traceIdInjectionPrefix	String	모든 문자열	로그에 트레이스 ID를 삽입하기 전에 제공된 접두사를 포함합니다.	N/A	N/A	없음 (빈 문자열)
samplingRulesManifest	String	절대 파일 경로	로컬 샘플링 전략에 대한 샘플링 규칙의 소스 또는 중앙 전략에 대한 대체 규칙으로 사용할 사용자 지정 샘플링 규칙 파일의 경로입니다.	N/A	N/A	DefaultSamplingRules.json

속성 이름	유형	유효값	설명	환경 변수	시스템 속성	Default
awsServiceHandlerManifest	String	절대 파일 경로	AWS SDK 클라이언트로부터 추가 정보를 캡처하는 사용자 지정 매개변수 허용 목록의 경로입니다.	N/A	N/A	DefaultOperationParameterWhitelist.json
awsSdkVersion	Integer	1, 2	사용 중인 Java용AWS SDK 버전입니다. `awsServiceHandlerManifest` 이 설정되지 않은 경우 무시됩니다.	N/A	N/A	2
maxStackTraceLength	정수	음수가 아닌 정수	트레이스에 기록할 스택 트레이스의 최대 줄 수입니다.	N/A	N/A	50

속성 이름	유형	유효값	설명	환경 변수	시스템 속성	Default
streamingThreshold	정수	음수가 아닌 정수	최소한 이만큼의 서브세그먼트가 닫힌 후에는 청크가 너무 커지는 것을 방지하기 위해 아웃오브밴드 (out-of-band) 대몬 (daemon) 으로 스트리밍됩니다.	N/A	N/A	100
traceIdInjection	불	True, False	로깅 구성에 설명된 종속성 및 구성도 추가된 경우 로그에 X-Ray trace ID 삽입을 활성화합니다. 그렇지 않으면 아무 작업도 수행하지 않습니다.	N/A	N/A	TRUE

속성 이름	유형	유효값	설명	환경 변수	시스템 속성	Default
pluginsEnabled	불	True, False	운영 중인 AWS 환경에 대한 메타데이터를 기록하는 플러그인을 활성화합니다. 플러그인을 참조하십시오.	N/A	N/A	TRUE
collectSqlQueries	불	True, False	SQL 쿼리 문자열을 SQL 하위 세그먼트에 최선을 다해 기록합니다.	N/A	N/A	FALSE

속성 이름	유형	유효값	설명	환경 변수	시스템 속성	Default
contextPropagation	불	True, False	true인 경우 스레드 간에 X-Ray 컨텍스트를 자동으로 전파합니다. 그렇지 않으면 ThreadLocal을 사용하여 컨텍스트를 저장하고 스레드 간 수동 전파가 필요합니다.	N/A	N/A	TRUE

로깅 구성

X-Ray 에이전트의 로그 수준은 Java용 X-Ray SDK와 동일한 방식으로 구성할 수 있습니다. Java용 X-Ray SDK를 사용하여 로깅을 구성하는 방법에 대한 자세한 내용은 로깅를 참조하십시오.

수동 구성

에이전트의 자동 계측 외에도 수동 계측을 수행하려는 경우 프로젝트에 종속 항목으로 X-Ray SDK를 추가하세요. 수신 요청 추적에서 언급한 SDK의 사용자 지정 서블릿 필터는 X-Ray 에이전트와 호환되지 않는다는 점에 유의하십시오.

> **Note**
>
> 에이전트를 사용하는 동시에 수동 계측을 수행하려면 최신 버전의 X-Ray SDK를 사용해야 합니다.

Maven 프로젝트에서 작업하는 경우 `pom.xml` 파일에 다음 종속성을 추가하세요.

```
<dependencies>
  <dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-xray-recorder-sdk-core</artifactId>
    <version>2.11.0</version>
  </dependency>
  </dependencies>
```

Gradle 프로젝트에서 작업하는 경우 `build.gradle` 파일에 다음 종속성을 추가하세요.

```
implementation 'com.amazonaws:aws-xray-recorder-sdk-core:2.11.0'
```

에이전트를 사용하는 동안 일반 SDK와 마찬가지로 주석, 메타데이터, 사용자 ID 외에도 사용자 지정 하위 세그먼트를 추가할 수 있습니다. 에이전트는 스레드 간에 컨텍스트를 자동으로 전파하므로 멀티 스레드 애플리케이션에서 작업할 때는 컨텍스트를 전파하기 위한 해결 방법이 필요하지 않습니다.

문제 해결

에이전트는 완전 자동 계측 기능을 제공하기 때문에 문제가 발생했을 때 문제의 근본 원인을 파악하기 어려울 수 있습니다. X-Ray 에이전트가 예상대로 작동하지 않는 경우 다음 문제와 해결 방법을 검토하세요. X-Ray 에이전트와 SDK는 자카르타 커먼즈 로깅 (JCL) 을 사용합니다. 로깅 출력을 보려면 다음 예제와 같이 JCL을 로깅 백엔드에 연결하는 브리지가 클래스 경로에 있는지 확인하십시오. `log4j-jcl` 또는 `jcl-over-slf4j`

문제: 애플리케이션에서 Java 에이전트를 활성화했지만 X-Ray 콘솔에 아무 것도 표시되지 않습니다.

X-Ray 대몬(daemon)이 동일한 시스템에서 실행되고 있습니까?

그렇지 않은 경우, X-Ray 대몬(daemon) 설명서를 참조하여 설정하십시오.

애플리케이션 로그에 “X-Ray 에이전트 레코더 초기화”와 같은 메시지가 표시됩니까?

에이전트를 애플리케이션에 올바르게 추가한 경우 이 메시지는 애플리케이션이 시작될 때 요청을 받기 시작하기 전에 INFO 레벨로 기록됩니다. 이 메시지가 없으면 Java 에이전트가 Java 프로세스와 함께 실행되고 있지 않은 것입니다. 모든 설정 단계를 오타 없이 올바로 따랐는지 확인하십시오.

애플리케이션 로그에 "경 AWS X-Ray 계 정보 누락 예외 표시 안 함"과 같은 오류 메시지가 여러 개 표시되나요?

이러한 오류는 에이전트가 AWS SDK 요청 또는 SQL 쿼리와 같은 다운스트림 요청을 계측하려고 하지만 에이전트가 세그먼트를 자동으로 생성할 수 없기 때문에 발생합니다. 이러한 오류가 많이 발생하는 경우 에이전트가 사용 사례에 가장 적합한 도구가 아닐 수 있으므로 대신 X-Ray SDK를 사용한 수동 계측을 고려해 볼 수 있습니다. 또는 X-Ray SDK 디버그 로그를 활성화하여 컨텍스트 누락 예외가 발생한 위치의 스택 추적을 볼 수 있습니다. 코드의 이러한 부분을 사용자 지정 세그먼트로 래핑하여 이러한 오류를 해결할 수 있습니다. 다운스트림 요청을 사용자 지정 세그먼트로 래핑하는 예제는 스타트업 코드 계측의 샘플 코드를 참조하십시오.

문제: 예상했던 일부 세그먼트가 X-Ray 콘솔에 표시되지 않습니다.

애플리케이션이 멀티스레딩을 사용하나요?

생성될 것으로 예상되는 일부 세그먼트가 콘솔에 표시되지 않는 경우 애플리케이션의 백그라운드 스레드가 원인일 수 있습니다. 애플리케이션이 AWS SDK를 사용하여 Lambda 함수에 대한 일회성 호출을 수행하거나 일부 HTTP 엔드포인트를 주기적으로 폴링하는 등 '방화 및 잊음'인 백그라운드 스레드를 사용하여 작업을 수행하는 경우, 스레드 간에 컨텍스트를 전파하는 동안 에이전트가 혼동될 수 있습니다. 이것이 문제인지 확인하려면 X-Ray SDK 디버그 로그를 활성화하고 진행 중인 하위 세그먼트의 상위 세그먼트로 이름이 지정된 세그먼트를 내보내지 않음과 <NAME >과 같은 메시지가 있는지 확인하세요. 이 문제를 해결하려면 서버가 돌아오기 전에 백그라운드 스레드를 결합하여 해당 스레드에서 수행한 모든 작업이 기록되도록 할 수 있습니다. 또는 백그라운드 스레드에서 컨텍스트 전파를 비활성화하도록 에이전트의 `contextPropagation` 구성을 `false`로 설정할 수 있습니다. 이렇게 하면 사용자 지정 세그먼트를 사용하여 해당 스레드를 수동으로 계측하거나 해당 스레드에서 발생하는 컨텍스트 누락 예외를 무시해야 합니다.

샘플링 규칙을 설정하셨나요?

X-Ray 콘솔에 무작위로 또는 예상치 못한 세그먼트가 표시되거나 콘솔에 표시될 것으로 예상한 세그먼트가 표시되지 않는 경우 샘플링 문제가 발생한 것일 수 있습니다. X-Ray 에이전트는 X-Ray 콘솔의 규칙을 사용하여 생성하는 모든 세그먼트에 중앙 집중식 샘플링을 적용합니다. 기본 규칙은 초당 1개의 세그먼트이며 이후 세그먼트의 5% 가 샘플링됩니다. 즉, 에이전트를 사용하여 빠르게 생성된 세그먼트는 샘플링되지 않을 수 있습니다. 이 문제를 해결하려면 X-Ray 콘솔에서 원하는 세그먼트를 적절히 샘플링하는 사용자 지정 샘플링 규칙을 만들어야 합니다. 자세한 내용은 샘플링을 참조하십시오.

Java용 X-Ray SDK 구성

Java용 X-Ray SDK에는 전역 레코더를 제공하는 `AWSXRay`라는 클래스가 있습니다. 이는 코드를 구성하는 데 사용할 수 있는 `TracingHandler`입니다. 전역 레코더를 구성하여 수신 HTTP 호출에 대해 세그먼트를 생성하는 `AWSXRayServletFilter`를 사용자 지정할 수 있습니다.

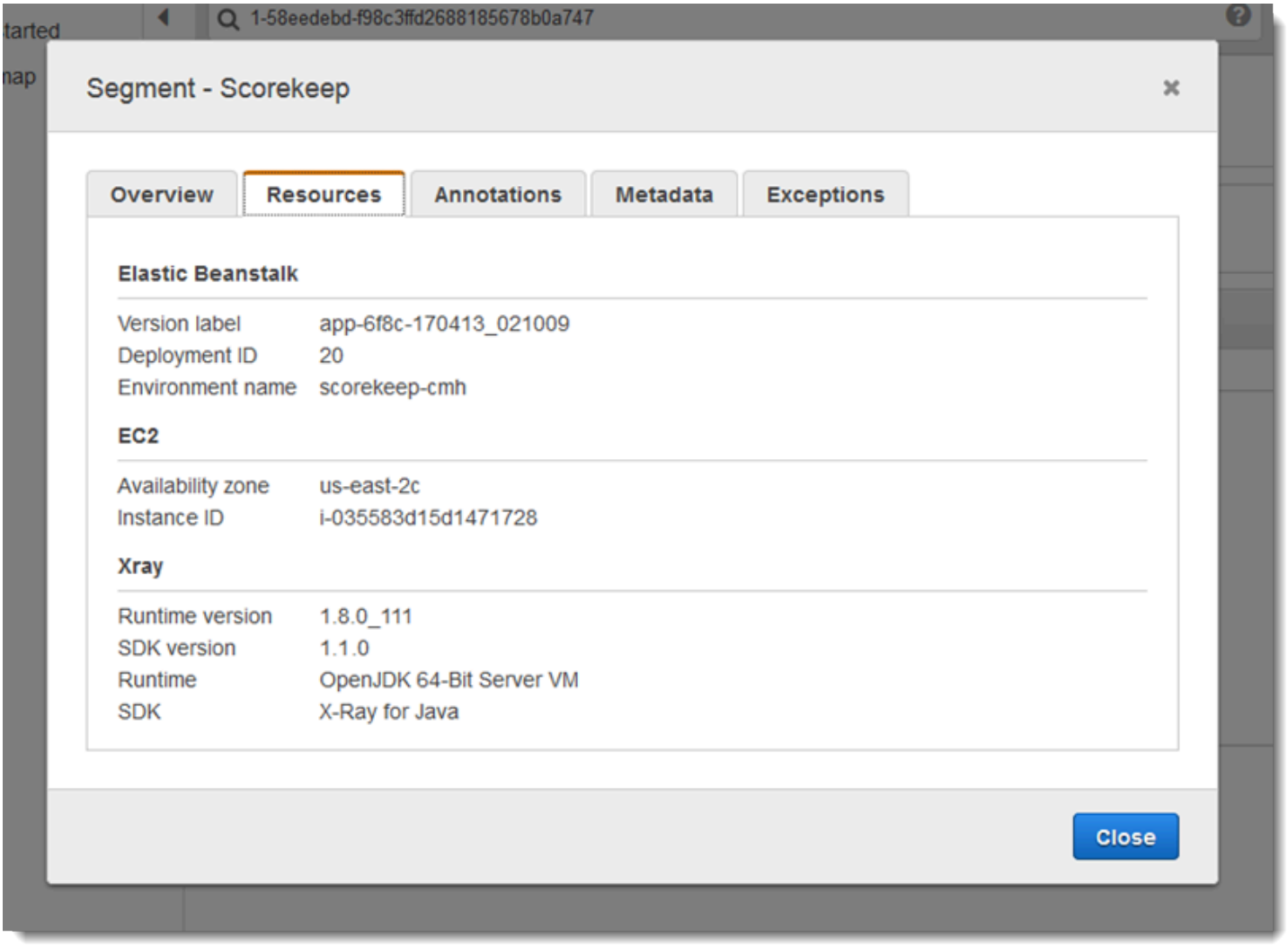
Sections

서비스 플러그인

`plugins`을 사용하여 애플리케이션을 호스팅하는 서비스에 대한 정보를 기록할 수 있습니다.

플러그인

- Amazon EC2 — `EC2Plugin`은 인스턴스 ID, 가용 영역 및 CloudWatch Logs 그룹을 추가합니다.
- Elastic Beanstalk – `ElasticBeanstalkPlugin`이 환경 이름, 버전 레이블 및 배포 ID를 추가합니다.
- Amazon ECS — `ECSPlugin`이 컨테이너 ID를 추가합니다.
- Amazon EKS – `EKSPlugin`은 컨테이너 ID, 클러스터 이름, 포드 ID 및 CloudWatch Logs 그룹을 추가합니다.



플러그인을 사용하려면 `AWSXRayRecorderBuilder`에서 `withPlugin`을 호출하십시오.

Example src/main/java/scorekeep/WebConfig.java - 레코더

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

SDK는 플러그인 설정을 사용하여 세그먼트에 `origin` 필드를 설정하기도 합니다. 이는 애플리케이션을 실행하는 AWS 리소스 유형을 나타냅니다. 여러 플러그인을 사용하는 경우 SDK는 ElasticBeanstalk> EKS> ECS> EC2 순서로 확인하여 오리진을 결정합니다.

샘플링 규칙

SDK는 X-Ray 콘솔에서 정의하는 샘플링 규칙을 사용하여 기록할 요청을 결정합니다. 기본 규칙은 매초 첫 번째 요청을 추적하고, 모든 서비스에서 추가 요청의 5%를 X-Ray로 추적 전송합니다. X-Ray 콘솔에서 추가 규칙을 생성하여 각 애플리케이션에 대해 기록되는 데이터의 양을 사용자 지정합니다.

SDK는 사용자 지정 규칙을 정의된 순서대로 적용합니다. 요청이 여러 사용자 지정 규칙과 일치하는 경우 SDK는 첫 번째 규칙만 적용합니다.

> **Note**
>
> SDK가 샘플링 규칙을 가져오기 위해 X-Ray에 연결할 수 없는 경우, 매초 첫 번째 요청과 호스트당 추가 요청의 5%에 대한 기본 로컬 규칙으로 되돌아갑니다. 호스트가 샘플링 API를 직접 호출할 수 있는 권한이 없거나, X-Ray 대몬(daemon)에 연결할 수 없을 경우 이러한 상황이 발생할 수 있고 이 대몬(daemon)은 SDK에서 수행한 API 직접 호출에 대한 TCP 프록시 역할을 합니다.

JSON 문서에서 샘플링 규칙을 불러오도록 SDK를 구성할 수도 있습니다. SDK는 X-Ray 샘플링을 사용할 수 없을 경우 로컬 규칙을 백업으로 사용하거나, 로컬 규칙을 전용으로 사용할 수 있습니다.

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
  }
}
```

이 예에서는 하나의 사용자 지정 규칙과 기본 규칙을 정의합니다. 사용자 지정 규칙은 최소 추적 요청 수 없이 5% 샘플링 비율을 `/api/move/` 아래 경로에 적용합니다. 기본 규칙은 매초 최초 요청과 추가 요청의 10%를 추적합니다.

로컬로 규칙의 정의할 때의 단점은 고정 대상이 X-Ray 서비스를 통해 관리되는 대신, 레코더의 각 인스턴스별로 독립적으로 적용된다는 것입니다. 호스트를 많이 배포할수록 고정 속도가 크게 증대하기 때문에 기록되는 데이터의 양을 제어하기가 어려워집니다.

에서는 샘플링 속도를 수정할 수 AWS Lambda없습니다. 구성된 서비스가 함수를 호출하는 경우 해당 서비스에서 샘플링한 요청을 생성한 호출이 Lambda에 의해 기록됩니다. 활성 추적이 활성화되고 트레이스 헤더가 없는 경우 Lambda에서 샘플링 결정을 내립니다.

Spring에서 백업 규칙을 제공하려면 구성 클래스에서 `CentralizedSamplingStrategy`를 사용하여 전역 레코더를 구성합니다.

Example src/main/java/myapp/WebConfig.java – 레코더 구성

```
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());
}
```

Tomcat의 경우 `ServletContextListener`를 확장하는 리스너를 추가하고 이 리스너를 배포 설명자에 등록합니다.

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>
```

로컬 규칙만 사용하려면 `CentralizedSamplingStrategy`를 `LocalizedSamplingStrategy`로 바꿉니다.

```
builder.withSamplingStrategy(new LocalizedSamplingStrategy(ruleFile));
```

로깅

기본적으로 이 SDK는 `ERROR` 수준 및 수준 메시지를 애플리케이션 로그로 출력합니다. SDK에 대한 디버그 수준 로깅을 활성화하여 보다 상세한 로그를 애플리케이션 로그 파일로 출력할 수 있습니다. 유효한 로그 수준은`DEBUG`, `INFO`, `WARN`, `ERROR`, 및 `FATAL`입니다. `FATAL` 로그 수준은 SDK가 치명적 수준에서 로깅하지 않기 때문에 모든 로그 메시지를 무음 처리합니다.

Example 애플리케이션 속성

`logging.level.com.amazonaws.xray` 속성을 사용하여 로깅 수준을 설정합니다.

```
logging.level.com.amazonaws.xray = DEBUG
```

디버그 로그를 사용하여 수동으로 하위 세그먼트를 생성할 때 미완료 하위 세그먼트와 같은 문제를 식별할 수 있습니다.

로그에 추적 ID 주입

현재 정규화된 트레이스 ID를 로그 문에 표시하려면 매핑된 진단 컨텍스트(MDC)에 해당 ID를 주입할 수 있습니다. 메서드는 `SegmentListener` 인터페이스를 사용하여 세그먼트 수명 주기 이벤트 중에 X-Ray 레코더에서 호출됩니다. 세그먼트 또는 하위 세그먼트가 시작되면 정규화된 트레이스 ID가 `AWS-XRAY-TRACE-ID` 키를 사용하여 MDC에 주입됩니다. 해당 세그먼트가 끝나면 MDC에서 키가 제거됩니다. 이렇게 하면 사용 중인 로깅 라이브러리에 트레이스 ID가 표시됩니다. 하위 세그먼트가 끝나면 부모 ID가 MDC에 주입됩니다.

Example 정규화된 트레이스 ID

정규화된 ID는 `TraceID@EntityID`로 표시됩니다.

```
1-5df42873-011e96598b447dfca814c156@541b3365be3dafc3
```

이 기능은 Java용 AWS X-Ray SDK로 구성된 Java 애플리케이션에서 작동하며 다음 로깅 구성을 지원합니다.

- Logback 백엔드가 있는 SLF4J 프런트 엔드 API
- Log4J2 백엔드가 있는 SLF4J 프런트 엔드 API
- Log4J2 백엔드가 있는 Log4J2 프런트 엔드 API

각 프런트 엔드와 각 백엔드의 요구 사항은 다음 탭을 참조하십시오.

SLF4J Frontend

1. 다음 Maven 종속성을 프로젝트에 추가합니다.

```
<dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-xray-recorder-sdk-slf4j</artifactId>
    <version>2.11.0</version>
</dependency>
```

2. `AWSXRayRecorder`를 구축할 때 `withSegmentListener` 메서드를 포함시킵니다. 이렇게 하면 새 트레이스 ID를 SLF4J MDC에 자동으로 주입하는 `SegmentListener` 클래스가 추가됩니다.

   `SegmentListener`는 선택적 문자열을 파라미터로 사용하여 log 문 접두사를 구성합니다. 접두사는 다음과 같은 방법으로 구성할 수 있습니다.

   - 없음 – 기본 `AWS-XRAY-TRACE-ID` 접두사를 사용합니다.
   - 비어 있음 – 빈 문자열(예: `""`)을 사용합니다.
   - 사용자 지정 – 문자열에 정의된 사용자 지정 접두사를 사용합니다.

   Example **`AWSXRayRecorderBuilder`** 명령문

```
AWSXRayRecorderBuilder builder = AWSXRayRecorderBuilder
        .standard().withSegmentListener(new SLF4JSegmentListener("CUSTOM-
PREFIX"));
```

Log4J2 front end

1. 다음 Maven 종속성을 프로젝트에 추가합니다.

```
<dependency>
```

```
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-xray-recorder-sdk-log4j</artifactId>
    <version>2.11.0</version>
</dependency>
```

2. AWSXRayRecorder를 구축할 때 withSegmentListener 메서드를 포함시킵니다. 이렇게 하면 새 정규화된 트레이스 ID를 SLF4J MDC에 자동으로 주입하는 SegmentListener 클래스가 추가됩니다.

   SegmentListener는 선택적 문자열을 파라미터로 사용하여 log 문 접두사를 구성합니다. 접두사는 다음과 같은 방법으로 구성할 수 있습니다.

   - 없음 – 기본 AWS-XRAY-TRACE-ID 접두사를 사용합니다.
   - 비어 있음 – 빈 문자열(예: "")을 사용하고 접두사를 제거합니다.
   - 사용자 지정 – 문자열에 정의된 사용자 지정 접두사를 사용합니다.

   Example **AWSXRayRecorderBuilder** 명령문

```
AWSXRayRecorderBuilder builder = AWSXRayRecorderBuilder
        .standard().withSegmentListener(new Log4JSegmentListener("CUSTOM-
PREFIX"));
```

Logback backend

트레이스 ID를 로그 이벤트에 삽입하려면 각 로깅 문의 형식을 지정하는 로거의 PatternLayout을 수정해야 합니다.

1. patternLayout이 구성된 위치를 찾습니다. 이 작업은 프로그래밍 방식으로 또는 XML 구성 파일을 통해 수행할 수 있습니다. 자세한 내용은 Logback 구성을 참조하십시오.
2. 이후 로깅 문에 트레이스 ID를 삽입하려면 patternLayout의 아무 곳에나 %X{AWS-XRAY-TRACE-ID}를 삽입합니다. %X{}는 MDC에서 제공된 키를 사용해 값을 검색하고 있음을 나타냅니다. Logback의 PatternLayout에 대한 자세한 내용은 PatternLayout을 참조하십시오.

Log4J2 backend

1. patternLayout이 구성된 위치를 찾습니다. 이 작업은 XML, JSON, YAML 또는 속성 형식으로 작성된 구성 파일을 통해 또는 프로그래밍 방식으로 수행할 수 있습니다.

구성 파일을 통해 Log4J2를 구성하는 방법에 대한 자세한 내용은 구성을 참조하십시오.

프로그래밍 방식으로 Log4J2를 구성하는 방법에 대한 자세한 내용은 프로그래밍 방식으로 구성을 참조하십시오.

2. 이후 로깅 문에 트레이스 ID를 삽입하려면 `PatternLayout`의 아무 곳에나 `%X{AWS-XRAY-TRACE-ID}`를 삽입합니다. `%X{}`는 MDC에서 제공된 키를 사용해 값을 검색하고 있음을 나타냅니다. Log4J2의 PatternLayout에 대한 자세한 내용은 PatternLayout을 참조하십시오.

트레이스 ID 주입 예제

다음은 트레이스 ID를 포함하도록 수정된 `PatternLayout` 문자열을 보여줍니다. 트레이스 ID는 스레드 이름(`%t`) 뒤에 및 로그 레벨(`%-5p`) 앞에 인쇄됩니다.

Example ID 주입이 있는 **`PatternLayout`**

```
%d{HH:mm:ss.SSS} [%t] %X{AWS-XRAY-TRACE-ID} %-5p %m%n
```

AWS X-Ray 는 쉽게 구문 분석할 수 있도록 로그 문에 키와 트레이스 ID를 자동으로 인쇄합니다. 다음은 수정된 `PatternLayout`을 사용하는 로그 문을 보여줍니다.

Example ID 주입이 있는 로그 문

```
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
```

로깅 메시지 자체는 패턴 `%m`에 보관되며 로거를 호출할 때 설정됩니다.

세그먼트 리스너

세그먼트 리스너는 `AWSXRayRecorder`에 의해 생성된 세그먼트의 시작 및 끝과 같은 수명 주기 이벤트를 가로챌 수 있는 인터페이스입니다. 세그먼트 리스너 이벤트 함수의 구현은 `onBeginSubsegment`로 생성될 때 모든 하위 세그먼트에 동일한 주석을 추가하거나, `afterEndSegment`를 사용하여 각 세그먼트를 데몬에 전송한 후 메시지를 기록하거나, `beforeEndSubsegment`를 사용하여 SQL 인터셉터가 전송한 쿼리를 기록하여 하위 세그먼트가 SQL 쿼리를 나타내는지 확인하고, 그렇다면 추가적인 메타데이터를 추가합니다.

전체 `SegmentListener` 함수 목록을 보려면 AWS X-Ray X-Ray Recorder SDK for Java API에 대한 설명서를 참조하십시오.

다음 예제에서는 onBeginSubsegment를 사용하여 생성할 때 모든 하위 세그먼트에 일관된 주석을 추가하고 afterEndSegment를 사용하여 각 세그먼트의 끝에 로그 메시지를 인쇄하는 방법을 보여줍니다.

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());
    }
}
```

이 사용자 지정 세그먼트 리스너는 AWSXRayRecorder를 빌드할 때 참조됩니다.

Example AWSXRayRecorderBuilder 문

```
AWSXRayRecorderBuilder builder = AWSXRayRecorderBuilder
        .standard().withSegmentListener(new MySegmentListener());
```

환경 변수

환경 변수를 사용하여 Java용 X-Ray SDK를 구성할 수 있습니다. SDK는 다음 변수를 지원합니다.

- AWS_XRAY_CONTEXT_MISSING – 열려 있는 세그먼트가 없는 경우 구성된 구성된 코드가 데이터를 기록하려고 할 때 발생하는 예외를 방지하려면 RUNTIME_ERROR로 설정합니다.

  유효한 값

  - RUNTIME_ERROR— 런타임 예외가 발생합니다.

  - `LOG_ERROR` – 오류를 기록하고 계속합니다 (기본값).
  - `IGNORE_ERROR`— 오류를 무시하고 계속합니다.

  열린 요청이 없을 때 실행되는 시작 코드 또는 새 스레드를 생성하는 코드에서 계측된 클라이언트를 사용하려고 하는 경우 누락된 세그먼트 또는 하위 세그먼트와 관련된 오류가 발생할 수 있습니다.
- `AWS_XRAY_DAEMON_ADDRESS` – X-Ray 대몬(daemon) 리스너의 호스트와 포트를 설정합니다. 기본적으로 SDK는 추적 데이터(UDP)와 샘플링(TCP) 모두에 `127.0.0.1:2000`을 사용합니다. 다른 포트에서 수신 대기하도록 대몬(daemon)을 구성한 경우 또는 다른 호스트에서 실행 중인 경우 이 변수를 사용합니다.

  형식

  - 동일한 포트 – *`address`*`:`*`port`*
  - 다른 포트 – `tcp:`*`address`*`:`*`port`* `udp:`*`address`*`:`*`port`*
- `AWS_LOG_GROUP` - 로그 그룹의 이름을 애플리케이션과 연결된 로그 그룹으로 설정합니다. 로그 그룹이 애플리케이션과 동일한 AWS 계정 및 리전을 사용하는 경우 X-Ray는이 지정된 로그 그룹을 사용하여 애플리케이션의 세그먼트 데이터를 자동으로 검색합니다. 로그 그룹에 대한 자세한 내용은 로그 그룹 및 스트림 작업을 참조하세요.
- `AWS_XRAY_TRACING_NAME` – SDK가 세그먼트에 사용할 서비스 이름을 설정합니다. 서블릿 필터의 세그먼트 이름 지정 전략에 설정한 서비스 이름을 재정의합니다.

환경 변수는 코드에 설정된 동등한 시스템 속성 및 값을 재정의합니다.

시스템 속성

시스템 속성을 환경 변수에 대한 JVM 지정 대체로 사용할 수 있습니다. SDK는 다음 속성을 지원합니다.

- `com.amazonaws.xray.strategy.tracingName` – `AWS_XRAY_TRACING_NAME`와 동등합니다.
- `com.amazonaws.xray.emitters.daemonAddress` – `AWS_XRAY_DAEMON_ADDRESS`와 동등합니다.
- `com.amazonaws.xray.strategy.contextMissingStrategy` – `AWS_XRAY_CONTEXT_MISSING`와 동등합니다.

시스템 속성과 환경 변수가 모두 설정될 경우 환경 변수 값이 사용됩니다. 어느 방법도 코드에 설정된 값을 재정의합니다.

Java용 X-Ray SDK로 수신 요청 추적하기

X-Ray SDK를 사용하여 애플리케이션이 Amazon EC2 AWS Elastic Beanstalk또는 Amazon ECS의 EC2 인스턴스에서 처리하는 수신 HTTP 요청을 추적할 수 있습니다.

`Filter`를 사용하여 수신 HTTP 요청을 구성합니다. 애플리케이션에 X-Ray 서블릿 필터를 추가하면 Java용 X-Ray SDK가 샘플링된 각 요청에 대해 세그먼트를 생성합니다. 이 세그먼트에는 HTTP 요청의 시간, 메서드 및 배치가 포함됩니다. 추가로 구성하면 이 세그먼트의 하위 세그먼트가 생성됩니다.

> **Note**
>
> AWS Lambda 함수의 경우 Lambda는 샘플링된 각 요청에 대해 세그먼트를 생성합니다. 자세한 내용은 AWS Lambda 그리고 AWS X-Ray 섹션을 참조하세요.

각 세그먼트에는 서비스 맵 안에서 애플리케이션을 식별하는 이름이 있습니다. 이 세그먼트의 이름이 정적으로 지정되도록 하거나, 수신 요청의 호스트 헤더를 기반으로 SDK가 동적으로 이름을 지정하도록 구성할 수 있습니다. 동적 이름 지정을 사용하면 요청의 도메인 이름에 따라 그룹을 추적하고 이름이 예상 패턴과 일치하지 않을 경우(예: 호스트 헤더가 위조된 경우) 기본 이름을 적용할 수 있습니다.

> **전달된 요청**
>
> 로드 밸런서 또는 기타 중개자가 애플리케이션으로 요청을 전달하는 경우, X-Ray는 IP 패킷 내 소스 IP가 아니라 요청의 `X-Forwarded-For` 헤더로부터 클라이언트 IP를 가져옵니다. 전달된 요청에 대해 기록된 클라어언트 IP는 위조될 수 있으므로 신뢰하면 안 됩니다.

요청이 전달되면 SDK는 세그먼트에 추가 필드를 설정하여 이를 나타냅니다. 세그먼트에 `x_forwarded_for`로 설정된 `true` 필드가 포함된 경우 클라이언트 IP는 HTTP 요청의 `X-Forwarded-For` 헤더로부터 가져옵니다.

메시지 핸들러는 다음 정보를 포함하는 `http` 블록을 이용해 각 수신 요청에 대한 세그먼트를 생성합니다.

- HTTP 메서드 – GET, POST, PUT, DELETE 등.
- 클라이언트 주소 – 요청을 전송한 클라이언트의 IP 주소.
- 응답 코드 – 완료된 요청의 HTTP 응답 코드.
- 시간 – 시작 시간(요청 수신) 및 종료 시간(응답 전송).

- 유저 에이전트 — 요청에서 가져온 `user-agent`입니다.
- 콘텐츠 길이 — 응답의 `content-length`입니다.

Sections

애플리케이션에 추적 필터 추가(Tomcat)

Tomcat의 경우, `<filter>`를 프로젝트의 `web.xml` 파일에 추가합니다. `fixedName` 파라미터로 수신 요청을 위해 생성된 세그먼트에 적용할 서비스 이름을 지정하십시오.

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>
```

애플리케이션에 추적 필터 추가(Spring)

Spring의 경우, `Filter`를 `WebConfig` 클래스에 추가합니다. 세그먼트 이름을 하나의 문자열로 `AWSXRayServletFilter` 생성자에 보냅니다.

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");
  }
}
```

세그먼트 이름 지정 전략 구성

AWS X-Ray 는 서비스 이름을 사용하여 애플리케이션을 식별하고 애플리케이션이 사용하는 다른 애플리케이션, 데이터베이스, 외부 APIs 및 AWS 리소스와 구분합니다. X-Ray SDK는 수신 요청에 대한 세그먼트를 생성할 때 해당 세그먼트의 이름 필드에 애플리케이션의 서비스 이름을 기록합니다.

X-Ray SDK는 HTTP 요청 헤더의 호스트 이름 뒤에 세그먼트를 지정할 수 있습니다. 그러나 이 헤더가 위조되면 서비스 맵에 예기치 않은 노드가 발생할 수 있습니다. 위조된 호스트 헤더가 포함된 요청으로 인해 SDK가 잘못된 세그먼트 이름을 지정하는 현상을 방지하려면 들어오는 요청의 기본 이름을 지정해야 합니다.

애플리케이션이 여러 도메인의 요청을 처리하는 경우, 동적 이름 지정 전략을 사용하여 이를 세그먼트 이름에 반영하도록 SDK를 구성할 수 있습니다. 동적 이름 지정 전략을 사용하면 SDK가 예상 패턴과 일치하는 요청에 호스트 이름을 사용하고, 그렇지 않은 요청에 기본 이름을 적용할 수 있습니다.

예를 들어, 하나의 애플리케이션이 세 개의 하위 도메인 (`www.example.com`, `api.example.com`, `static.example.com`) 에 요청을 전송할 수 있습니다. `*.example.com` 패턴으로 동적 이름 지정 전략을 사용하여 각 하위도메인의 세그먼트를 서로 다른 이름으로 표시하면 서비스 맵에 서비스 노드가 세 개 생깁니다. 이 패턴에 맞지 않는 호스트 이름의 요청이 애플리케이션에 수신되면, 사용자가 지정한 대체 이름의 네 번째 노드가 서비스 맵에 표시됩니다.

모든 요청 세그먼트에 같은 이름을 사용하려면, 이전 단원에 나온 것처럼 서블릿 필터를 초기화할 때 애플리케이션 이름을 지정하십시오. 이것은 `SegmentNamingStrategy.fixed()`를 호출하고 `AWSXRayServletFilter` 생성자에 전달하여 SegmentNamingStrategy를 생성하는 것과 동일한 효과가 있습니다.

> **Note**
>
> 코드에 정의한 기본 서비스 이름을 `AWS_XRAY_TRACING_NAME` 환경 변수를 사용하여 재정의할 수 있습니다.

동적 이름 지정 전략은 호스트 이름이 일치해야 하는 패턴 및 HTTP 요청의 호스트 이름이 패턴과 일치하지 않는 경우 사용할 기본 이름을 정의합니다. Tomcat에서 세그먼트 이름을 동적으로 지정하려면, `dynamicNamingRecognizedHosts`와 `dynamicNamingFallbackName`을 이용해 패턴과 기본 이름을 따로 정의하십시오.

Example WEB-INF/web.xml – 동적 이름 지정을 이용하는 서블릿 필터

```
<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>
```

Spring의 경우, `SegmentNamingStrategy.dynamic()`를 호출하여 동적 SegmentNamingStrategy을 만들고 `AWSXRayServletFilter` 생성자로 보내십시오.

Example src/main/java/myapp/WebConfig.java – 동적 이름 지정을 이용하는 서블릿 필터

```
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"));
```

```
  }
}
```

Java용 X-Ray AWS SDK를 사용하여 SDK 호출 추적

애플리케이션이를 호출 AWS 서비스 하여 데이터를 저장하거나, 대기열에 쓰거나, 알림을 보내면 Java용 X-Ray SDK는 하위 세그먼트에서 다운스트림으로 호출을 추적합니다. 추적된 AWS 서비스 와 해당 서비스(예: Amazon S3 버킷 또는 Amazon SQS 대기열) 내에서 액세스하는 리소스는 X-Ray 콘솔의 트레이스 맵에 다운스트림 노드로 표시됩니다.

빌드에서 `aws-sdk` 및 `aws-sdk-instrumentor` 하위 모듈을 포함하면 Java용 X-Ray SDK가 모든 AWS SDK 클라이언트를 자동으로 계측합니다. Instrumentor 하위 모듈을 포함하지 않을 경우 다른 클라이언트는 배제하고 일부 클라이언트만 선택하여 구성할 수 있습니다.

개별 클라이언트를 계측하려면 빌드에서 `aws-sdk-instrumentor` 하위 모듈을 제거하고 서비스의 클라이언트 빌더를 사용하여 AWS SDK 클라이언트`TracingHandler`에를 `XRayClient`로 추가합니다.

예를 들어 `AmazonDynamoDB` 클라이언트를 구성하려면 트레이스 핸들러를`AmazonDynamoDBClientBuilder`로 전달합니다.

Example MyModel.java - DynamoDB 클라이언트

```
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();
...
```

모든 서비스의 경우, X-Ray 콘솔에서 호출된 API의 이름을 볼 수 있습니다. 서비스 하위 집합에 대해서는 X-Ray SDK가 세그먼트에 정보를 추가하여 서비스 맵에서 추가 세분화를 제공합니다.

예를 들어 계측된 DynamoDB 클라이언트에서 직접 호출을 생성하는 경우 SDK가 특정 테이블을 대상으로 한 직접 호출에 대해 테이블 이름을 세그먼트에 추가합니다. 콘솔에서, 각 테이블이 개별 노드로 서비스 맵에 표시되고, 특정 테이블을 대상으로 하지 않은 직접 호출에 대해 일반 DynamoDB 노드가 표시됩니다.

Example 항목을 저장하기 위한 DynamoDB 직접 호출에 대한 하위 세그먼트

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

명명된 리소스에 액세스할 때 다음 서비스를 호출할 경우 서비스 맵에 추가 노드가 생성됩니다. 특정 리소스를 대상으로 하지 않는 경우 서비스를 직접 호출하면 해당 서비스에 대한 일반 노드가 생성됩니다.

- Amazon DynamoDB – 테이블 이름
- Amazon Simple Storage Service –버킷 및 키 이름
- Amazon Simple Queue Service – 대기열 이름

AWS SDK for Java 2.2 이상을 AWS 서비스 사용하여에 대한 다운스트림 호출을 계측하려면 빌드 구성에서 `aws-xray-recorder-sdk-aws-sdk-v2-instrumentor` 모듈을 생략하면 됩니다. 대신 `aws-xray-recorder-sdk-aws-sdk-v2 module`을 포함한 다음, `TracingInterceptor`로 구성하여 개별 클라이언트를 계측합니다.

Example AWS SDK for Java 2.2 이상 - 추적 인터셉터

```
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();
//...
```

Java용 X-Ray SDK를 사용하여 다운스트림 HTTP 웹 서비스에 대한 호출 추적하기

애플리케이션이 마이크로서비스 또는 공개 HTTP API를 호출할 때 Java용 X-Ray SDK 버전의 `HttpClient`를 사용하여 해당 호출을 계측하고 API를 다운스트림 서비스로 서비스 그래프에 추가할 수 있습니다.

Java용 X-Ray SDK는 발신 HTTP 호출을 계측하기 위해 Apache HttpComponents에 상응하는 클래스 대신 사용할 수 있는 `DefaultHttpClient` 및 `HttpClientBuilder` 클래스를 포함합니다.

- `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`

이러한 라이브러리는 [aws-xray-recorder-sdk-apache-http] 하위 모듈에 있습니다.

모든 클라이언트를 계측하기 위해 기존 가져오기 문을 X-Ray에 해당하는 것으로 대체하거나, 특정 클라이언트를 계측하기 위해 클라이언트를 초기화할 때 정규화된 이름을 사용할 수 있습니다.

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();
    }
  }
```

다운스트림 웹 API에 대한 직접 호출을 계측할 때 Java용 X-Ray SDK가 HTTP 요청 및 응답에 대한 정보가 포함된 하위 세그먼트를 기록합니다. X-Ray는 하위 세그먼트를 사용하여 원격 API에 대해 추정된 세그먼트를 생성합니다.

Example 다운스트림 HTTP 호출에 대한 하위 세그먼트

```
{
  "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 다운스트림 HTTP 호출에 대한 추정된 세그먼트

```
{
```

```
  "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
}
```

Java용 X-Ray SDK로 SQL 쿼리 추적하기

SQL 인터셉터

Java용 X-Ray SDK JDBC 인터셉터를 데이터 소스 구성에 추가해 SQL 데이터베이스 쿼리를 계측하십시오.

- PostgreSQL – `com.amazonaws.xray.sql.postgres.TracingInterceptor`
- MySQL – `com.amazonaws.xray.sql.mysql.TracingInterceptor`

이러한 인터셉터는 각각 aws-xray-recorder-sql-postgres 및 aws-xray-recorder-sql-mysql 하위 모듈에 있습니다. 인터셉터는 `org.apache.tomcat.jdbc.pool.JdbcInterceptor`를 구현하며 Tomcat 연결 풀과 호환됩니다.

> **Note**
>
> SQL 인터셉터는 보안을 위해 하위 세그먼트 내에서 SQL 쿼리 자체를 기록하지 않습니다.

Spring인 경우에는 인터셉터를 속성 파일에 추가하고 Spring Boot의 `DataSourceBuilder`로 데이터 소스를 구축하십시오.

Example **src/main/java/resources/application.properties** - PostgreSQL JDBC 인터셉터

```
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** - 데이터 소스

```
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();
  }
...
}
```

Tomcat의 경우에는 JDBC 데이터 소스에서 Java용 X-Ray SDK 클래스를 참조하여 `setJdbcInterceptors`를 직접 호출합니다.

Example **`src/main/myapp/model.java`** - 데이터 소스

```
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 데이터 소스 라이브러리는 Java용 X-Ray SDK에 포함되지만, 사용하는 문서에 제공된 종속성으로 이 라이브러리를 선언할 수 있습니다.

Example **`pom.xml`** - JDBC 데이터 소스

```
<dependency>
  <groupId>org.apache.tomcat</groupId>
  <artifactId>tomcat-jdbc</artifactId>
  <version>8.0.36</version>
  <scope>provided</scope>
</dependency>
```

네이티브 SQL 트레이싱 데코레이터

- aws-xray-recorder-sdk-sql을 종속성에 추가합니다.
- 데이터베이스 데이터 소스, 연결 또는 명령문을 꾸며보세요.

```
dataSource = TracingDataSource.decorate(dataSource)
connection = TracingConnection.decorate(connection)
statement = TracingStatement.decorateStatement(statement)
preparedStatement = TracingStatement.decoratePreparedStatement(preparedStatement,
 sql)
callableStatement = TracingStatement.decorateCallableStatement(callableStatement,
 sql)
```

Java용 X-Ray SDK를 사용하여 사용자 지정 하위 세그먼트 생성하기

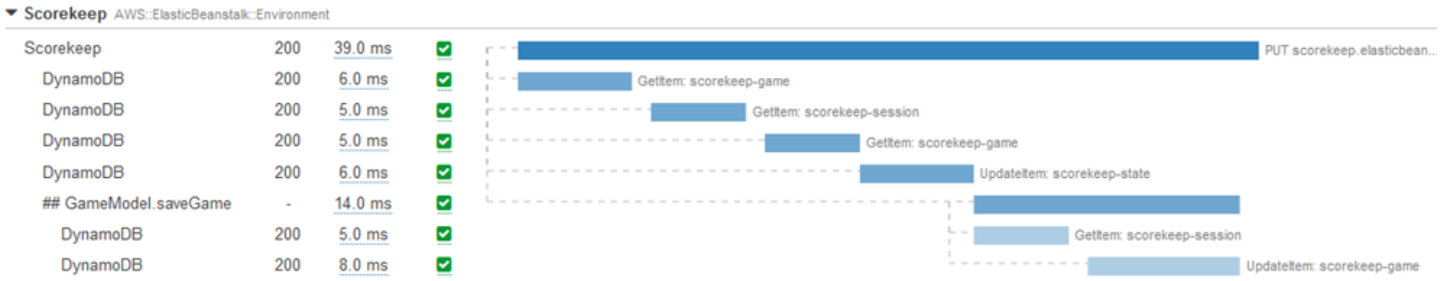
하위 세그먼트는 추적의 세그먼트를 확장하여 요청을 처리하기 위해 완료된 작업에 대한 세부 정보를 표시합니다. 계측되는 클라이언트에서 직접 호출할 때마다, X-Ray SDK는 하위 세그먼트 안에 생성된 정보를 기록합니다. 추가 하위 세그먼트를 생성하여 다른 하위 세그먼트를 그룹화하거나, 코드 섹션의 성능을 평가하거나, 주석 및 메타데이터를 기록할 수 있습니다.

하위 세그먼트를 관리하려면 `beginSubsegment` 및 `endSubsegment` 메서드를 사용합니다.

Example GameModel.java – 사용자 지정 하위 세그먼트

```
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();
    }
  }
```

이 예제에서는 하위 세그먼트 내의 코드가 세션 모델의 메서드를 사용하여 DynamoDB에서 게임의 세션을 로드하고 AWS SDK for Java의 DynamoDB 매퍼를 사용하여 게임을 저장합니다. 하위 세그먼트에서 이 코드를 래핑하면 콘솔의 추적 보기에서 `Save Game` 하위 세그먼트의 DynamoDB 하위가 호출됩니다.



하위 세그먼트 내 코드가 확인된 예외를 내보낼 경우 하위 세그먼트가 항상 닫히도록 코드를 `try` 블록에 래핑하고 `finally` 블록에서 `AWSXRay.endSubsegment()`를 호출합니다. 하위 세그먼트가 닫히지 않는 경우 상위 세그먼트가 완료될 수 없으며 X-Ray로 전송되지 않습니다.

확인된 예외를 내보내지 않는 코드의 경우 코드를 Lambda 함수로 `AWSXRay.CreateSubsegment`에 전달할 수 있습니다.

Example 하위 세그먼트 Lambda 함수

```
import com.amazonaws.xray.AWSXRay;

AWSXRay.createSubsegment("getMovies", (subsegment) -> {
    // function code
});
```

하위 세그먼트를 세그먼트 또는 다른 하위 세그먼트 내에서 생성하면 Java용 X-Ray SDK가 해당 하위 세그먼트에 대해 ID를 생성하고 시작 시간 및 종료 시간을 기록합니다.

Example 메타데이터가 포함된 하위 세그먼트

```
"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"
    }
  },
```

비동기 및 다중 스레드 프로그래밍의 경우 비동기 실행 중에 X-Ray 컨텍스트가 수정될 수 있으므로 하위 세그먼트를 `endSubsegment()` 메서드에 수동으로 전달하여 올바르게 닫히도록 해야 합니다. 상

위 세그먼트가 닫힌 후 비동기 하위 세그먼트가 닫히면, 이 메서드는 전체 세그먼트를 자동으로 X-Ray 대몬(daemon)으로 스트리밍합니다.

Example 비동기 서브세그먼트

```
@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();
}
```

Java용 X-Ray SDK로 세그먼트에 주석 및 메타데이터 추가하기

주석 및 메타데이터와 함께 요청, 환경 또는 애플리케이션에 대한 추가 정보를 기록할 수 있습니다. X-Ray SDK에서 생성하는 세그먼트 또는 사용자가 생성하는 사용자 지정 하위 세그먼트에 주석 및 메타데이터를 추가할 수 있습니다.

주석은 문자열, 숫자 또는 부울 값과 결합한 키-값 페어입니다. 주석은 필터 표현식에서 사용하기 위해 인덱싱됩니다. 주석은 콘솔의 트레이스를 그룹화할 때 사용할 데이터를 기록하거나 `GetTraceSummaries` API를 직접 호출할 때 사용하세요.

메타데이터는 객체 및 목록을 포함한 모든 유형의 값을 가질 수 있는 키-값 페어지만, 필터 표현식에 사용할 수 있도록 인덱싱되지는 않습니다. 트레이스에 저장하고 싶지만 검색에는 사용하지 않을 추가 데이터는 메타데이터를 사용하여 기록하십시오.

세그먼트에는 주석과 메타데이터 외에 사용자 ID 문자열도 기록할 수 있습니다. 사용자 ID는 세그먼트의 별도 필드에 기록되면 검색용으로 인덱스되지 않습니다.

Sections

- Java용 X-Ray SDK로 주석 기록하기

Java용 X-Ray SDK로 주석 기록하기

주석을 사용하여 검색용으로 인덱싱할 정보를 세그먼트나 하위 세그먼트에 기록하십시오.

주석 요구 사항

- 키 - X-Ray 주석의 키는 최대 500자의 영숫자를 포함할 수 있습니다. 점이나 마침표(.) 이외의 공백이나 기호를 사용할 수 없습니다.
- 값 - X-Ray 주석의 값은 최대 1,000자의 유니코드 문자를 포함할 수 있습니다.
- 주석 수 - 트레이스당 최대 50개의 주석을 사용할 수 있습니다.

주석 기록 방법

1. AWSXRay에서 현재 세그먼트나 하위 세그먼트의 참조를 가져오십시오.

```
import com.amazonaws.xray.AWSXRay;
import com.amazonaws.xray.entities.Segment;
...
Segment document = AWSXRay.getCurrentSegment();
```

or

```
import com.amazonaws.xray.AWSXRay;
import com.amazonaws.xray.entities.Subsegment;
...
Subsegment document = AWSXRay.getCurrentSubsegment();
```

2. 문자열 키, 부울, 숫자 또는 문자열 값으로 putAnnotation을 직접 호출합니다.

```
document.putAnnotation("mykey", "my value");
```

다음 예에서는 점이 포함된 문자열 키와 부울, 숫자 또는 문자열 값을 사용하여 putAnnotation을 직접 호출하는 방법을 보여줍니다.

```
document.putAnnotation("testkey.test", "my value");
```

SDK는 세그먼트 문서의 annotations 객체에 주석을 키-값 페어로 기록합니다. 같은 키로 putAnnotation을 두 번 직접 호출하면 같은 세그먼트나 하위 세그먼트에 기록했던 값을 덮어씁니다.

특정 값을 포함한 주석이 있는 트레이스를 찾으려면 annotation[*key*] 키워드를 필터 표현식에 사용하십시오.

Example **src/main/java/scorekeep/GameModel.java** – 주석 및 메타데이터

```
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();
    }
  }
```

Java용 X-Ray SDK로 메타데이터 기록하기

메타데이터를 이용해 검색용으로 인덱싱하지 않아도 되는 정보를 세그먼트나 하위 세그먼트에 기록하십시오. 메타데이터 값은 문자열, 숫자, 부울 또는 JSON 객체나 어레이에 직렬화할 수 있는 모든 객체가 될 수 있습니다.

메타데이터 기록 방법

1. AWSXRay에서 현재 세그먼트나 하위 세그먼트의 참조를 가져오십시오.

```
import com.amazonaws.xray.AWSXRay;
import com.amazonaws.xray.entities.Segment;
...
Segment document = AWSXRay.getCurrentSegment();
```

   or

```
import com.amazonaws.xray.AWSXRay;
import com.amazonaws.xray.entities.Subsegment;
...
Subsegment document = AWSXRay.getCurrentSubsegment();
```

2. 문자열 네임스페이스, 문자열 키 및 부울, 숫자, 문자열 또는 객체 값으로 putMetadata를 호출합니다.

```
document.putMetadata("my namespace", "my key", "my value");
```

   or

   키와 값만 이용해 putMetadata를 직접 호출합니다.

```
document.putMetadata("my key", "my value");
```

네임스페이스를 지정하지 않으면, SDK는 default를 사용합니다. 같은 키로 putMetadata을 두 번 직접 호출하면 같은 세그먼트나 하위 세그먼트에 기록했던 값을 덮어씁니다.

Example **src/main/java/scorekeep/GameModel.java** – 주석 및 메타데이터

```
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();
    }
  }
```

Java용 X-Ray SDK로 사용자 ID 기록하기

사용자 ID를 요청 세그먼트에 기록하여 요청을 보낸 사용자를 식별합니다.

사용자 ID 기록 방법

1. AWSXRay에서 현재 세그먼트에 대한 참조를 가져옵니다.

   ```
   import com.amazonaws.xray.AWSXRay;
   import com.amazonaws.xray.entities.Segment;
   ...
   Segment document = AWSXRay.getCurrentSegment();
   ```

2. 요청을 보낸 사용자의 문자열 ID로 setUser를 직접 호출합니다.

   ```
   document.setUser("U12345");
   ```

컨트롤러에서 setUser를 직접 호출하면 애플리케이션이 요청을 처리하는 순간부터 사용자 ID를 기록할 수 있습니다. 사용자 ID 설정용으로만 세그먼트를 사용한다면, 호출을 1줄로 연결할 수 있습니다.

Example src/main/java/scorekeep/MoveController.java – 사용자 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);
 }
```

사용자 ID의 트레이스를 찾으려면, `user` 키워드를 필터 표현식에 적용하십시오.

AWS X-Ray Java용 X-Ray SDK에 대한 지표

이 주제에서는 AWS X-Ray 네임스페이스, 지표 및 차원에 대해 설명합니다. Java용 X-Ray SDK를 사용하여 수집된 X-Ray 세그먼트에서 샘플링되지 않은 Amazon CloudWatch 지표를 게시할 수 있습니다. 이러한 지표는 세그먼트의 시작 및 종료 시간과 오류, 장애 및 스로틀된 상태 플래그에서 파생됩니다. 이러한 트레이스 지표를 사용하여 하위 세그먼트 내에서 재시도 및 종속성 문제를 표시할 수 있습니다.

CloudWatch는 지표 리포지토리입니다. 지표는 CloudWatch의 기본 개념으로 시간별로 정렬된 데이터 요소 세트를 나타냅니다. 사용자는 (또는 AWS 서비스) 지표 데이터 포인트를 CloudWatch에 게시하고 해당 데이터 포인트에 대한 통계를 정렬된 시계열 데이터 세트로 검색합니다.

지표는 이름, 네임스페이스 및 하나 이상의 차원으로 고유하게 정의됩니다. 각 데이터 포인트에는 타임스탬프가 있으며 선택 사항으로 측정 단위가 있습니다. 통계를 요청하면 네임스페이스, 지표 이름 및 차원으로 반환된 데이터 스트림이 식별됩니다.

CloudWatch에 대한 자세한 내용은 Amazon CloudWatch 사용 설명서를 참조하세요.

X-Ray CloudWatch 지표

`ServiceMetrics/SDK` 네임스페이스에는 다음과 같은 지표가 포함됩니다.

지표	사용 가능 통계	설명	단위
`Latency`	평균, 최소, 최대, 개수	시작 시간과 종료 시간 간의 차이입니다. 평균, 최소 및 최대 항목은 모두 작업 지연 시	밀리초

지표	사용 가능 통계	설명	단위
		간을 나타냅니다. 개수는 호출 수를 나타냅니다.	
ErrorRate	평균, 합계	4xx Client Error 상태 코드로 실패하여 오류가 발생한 요청 비율입니다.	%
FaultRate	평균, 합계	5xx Server Error 상태 코드로 실패하여 오류가 발생한 트레이스 비율입니다.	%
ThrottleRate	평균, 합계	429 상태 코드를 반환하는 스로틀된 트레이스 비율입니다. 이는 ErrorRate 지표의 하위 집합입니다.	%
OkRate	평균, 합계	OK 상태 코드를 발생시킨 트레이스된 요청 비율입니다.	%

X-Ray CloudWatch 차원

다음 표의 차원을 사용하여 X-Ray 계측 Java 애플리케이션에 대해 반환되는 지표를 구체화할 수 있습니다.

차원	설명
ServiceType	서비스 유형 (예: AWS::EC2::Instance 또는 알 수 없는 경우 NONE)입니다.
ServiceName	서비스의 정식 이름입니다.

X-Ray CloudWatch 지표 활성화하기

계측된 Java 애플리케이션에서 트레이스 지표를 활성화하려면 다음 절차를 따릅니다.

트레이스 지표를 구성하려면

1. `aws-xray-recorder-sdk-metrics` 패키지를 Apache Maven 종속성으로 추가합니다. 자세한 내용은 Java 서브모듈용 X-Ray SDK를 참조하세요.
2. 전역 레코더 빌드의 일부로서 새 `MetricsSegmentListener()`를 활성화합니다.

   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. CloudWatch 에이전트를 배포하여 Amazon Elastic Compute Cloud(Amazon EC2), Amazon Elastic Container Service(Amazon ECS) 또는 Amazon Elastic Kubernetes Service(Amazon EKS)를 사용하여 지표를 수집하세요:

   - Amazon EC2를 구성하려면 CloudWatch 에이전트 설치를 참조하세요.

- Amazon ECS를 구성하려면 Monitor Amazon ECS containers using Container Insights를 참조하세요.
- Amazon EKS를 구성하려면 Amazon CloudWatch Observability EKS 추가 기능을 사용하여 CloudWatch 에이전트 설치를 참조하세요.

4. CloudWatch 에이전트와 통신하도록 SDK를 구성합니다. 기본적으로 SDK는 `127.0.0.1` 주소의 CloudWatch 에이전트와 통신합니다. 환경 변수 또는 Java 속성을 `address:port`로 설정하여 대체 주소를 구성할 수 있습니다.

   Example 환경 변수

   ```
   AWS_XRAY_METRICS_DAEMON_ADDRESS=address:port
   ```

   Example Java 속성

   ```
   com.amazonaws.xray.metrics.daemonAddress=address:port
   ```

구성을 확인하려면

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/cloudwatch/:// https://https://://https://://CloudWatch://://https://://https://://https://://https://https://://https://://https
2. 지표 탭을 열어 지표의 유입을 관찰합니다.
3. (선택 사항) CloudWatch 콘솔의 로그 탭에서 `ServiceMetricsSDK` 로그 그룹을 엽니다. 호스트 지표와 일치하는 로그 스트림을 찾고 로그 메시지를 확인합니다.

멀티스레드 애플리케이션에서 스레드 간 세그먼트 컨텍스트 전달

애플리케이션에서 새 스레드를 생성하는 경우 `AWSXRayRecorder`가 현재 세그먼트 또는 하위 세그먼트 개체에 대한 참조를 유지하지 않습니다. 새 스레드에서 구성된 클라이언트를 사용하는 경우 SDK는 존재하지 않는 세그먼트에 쓰려고 하기 때문에 SegmentNotFoundException이 발생합니다.

개발하는 동안 예외 발생을 방지하려면 대신 오류를 로그할 것을 알리는 ContextMissingStrategy로 레코더를 구성합니다. SetContextMissingStrategy를 사용하여 코드에서 전략을 구성하거나 환경 변수 또는 시스템 속성을 사용하여 동등한 수준의 옵션을 구성할 수 있습니다.

오류를 처리하는 하나의 방법은 스레드를 시작할 때 beginSegment를 호출하고 스레드를 닫을 때 endSegment를 호출하여 새 세그먼트를 사용하는 것입니다. 이는 애플리케이션이 시작할 때 실행하는 코드와 같이 HTTP 요청에 대한 응답에서 실행하지 않는 코드를 구성하는 경우 유용합니다.

여러 스레드를 사용하여 수신 요청을 처리하는 경우 현재 세그먼트 또는 하위 세그먼트를 새 스레드에 전달하고 이를 전역 레코더에 제공할 수 있습니다. 이렇게 하면 새 스레드 내에서 레코딩된 정보가 해당 요청에 대해 레코딩된 나머지 정보와 동일한 세그먼트와 연결됩니다. 새 스레드에서 세그먼트를 사용할 수 있게 되면 `segment.run(() -> { ... })` 메서드를 사용하여 해당 세그먼트의 컨텍스트에 대한 액세스 권한이 있는 실행 가능 항목을 실행할 수 있습니다.

예제는 작업자 스레드에서 구성된 클라이언트 사용 섹션을 참조하세요.

비동기 프로그래밍과 함께 X-Ray 사용하기

Java용 X-Ray SDK는 SegmentContextExecutors를 사용하여 비동기 Java 프로그램에서 사용할 수 있습니다. SegmentContextExecutor는 실행자 인터페이스를 구현하므로, CompletableFuture의 모든 비동기 작업에 전달될 수 있습니다. 이렇게 하면 모든 비동기 작업이 해당 컨텍스트에서 올바른 세그먼트를 사용하여 실행될 수 있습니다.

Example 예제 App.java: SegmentContextExecutor를 CompletableFuture에 전달하기

```
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());
```

Spring 및 Java용 X-Ray SDK를 사용한 AOP

이 주제에서는 X-Ray SDK와 Spring 프레임워크를 사용하여 핵심 로직을 변경하지 않고 애플리케이션을 구성하는 방법을 설명합니다. 즉, 이제 원격으로 실행되는 애플리케이션을 계측하는 비침습적 방법이 있습니다 AWS.

Spring에서 AOP를 활성화하는 방법

1. Spring 구성
2. 애플리케이션에 트레이싱 필터 추가
3. 코드에 주석 추가 또는 인터페이스 구현
4. 애플리케이션에서 X-Ray 활성화

Spring 구성하기

Maven 또는 Gradle을 사용하면 AOP를 사용해 애플리케이션을 도구화하도록 Spring을 구성할 수 있습니다.

Maven을 사용하여 애플리케이션을 빌드하는 경우 `pom.xml` 파일에서 다음 종속성을 추가합니다.

```
<dependency>
     <groupId>com.amazonaws</groupId>
     <artifactId>aws-xray-recorder-sdk-spring</artifactId>
     <version>2.11.0</version>
</dependency>
```

Gradle의 경우 `build.gradle` 파일에 다음 종속성을 추가합니다.

```
compile 'com.amazonaws:aws-xray-recorder-sdk-spring:2.11.0'
```

Spring Boot 구성하기

이전 섹션에서 설명한 Spring 종속성 외에도 Spring Boot를 사용하는 경우 클래스 경로에 아직 없는 경우 다음 종속성을 추가하세요.

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'
```

애플리케이션에 트레이싱 필터 추가

WebConfig 클래스에 Filter를 추가하세요. 세그먼트 이름을 하나의 문자열로 AWSXRayServletFilter 생성자에 보냅니다. 추적 필터 및 수신 요청 계측 방법에 대한 자세한 내용은 Java용 X-Ray SDK로 수신 요청 추적하기을 참조하세요.

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");
  }
}
```

자카르타 지원

Spring 6는 엔터프라이즈 에디션에 Javax 대신 자카르타를 사용합니다. 이 새로운 네임스페이스를 지원하기 위해 X-Ray는 자체 자카르타 네임스페이스에 있는 병렬 클래스 집합을 만들었습니다.

필터 클래스의 경우 javax을 jakarta로 바꾸십시오. 세그먼트 네이밍 전략을 구성할 때는 다음 예시와 같이 네이밍 전략 클래스 이름 앞에 jakarta을 추가합니다:

```
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"));
    }
}
```

코드에 주석 추가 또는 인터페이스 구현

클래스는 @XRayEnabled 어노테이션으로 주석을 달거나 XRayTraced 인터페이스를 구현해야 합니다. 이는 AOP 시스템에 X-Ray 구성을 위해 영향을 받는 클래스의 함수를 래핑하도록 지시합니다.

애플리케이션에서 X-Ray 활성화하기

애플리케이션에서 X-Ray 트레이싱을 활성화하려면 코드에서 다음 메서드를 재정의하여 추상 클래스 BaseAbstractXRayInterceptor를 확장해야 합니다.

- generateMetadata—이 함수를 사용하면 현재 함수의 트레이스에 첨부된 메타데이터를 사용자 지정할 수 있습니다. 기본적으로 실행 함수의 클래스 이름은 메타데이터에 기록됩니다. 추가 정보가 필요한 경우 데이터를 더 추가할 수 있습니다.
- xrayEnabledClasses—이 함수는 비어 있으며 그대로 유지해야 합니다. 이 함수는 인터셉터에게 래핑할 메서드를 지시하는 Pointcut에 대해 호스트의 역할을 합니다. @XRayEnabled 주석이 추가된 클래스 중 트레이스할 클래스를 지정하여 Pointcut을 정의합니다. 다음 Pointcut 문은 인터셉터에 @XRayEnabled 주석이 추가된 모든 컨트롤러 빈을 래핑하도록 지시합니다.

  ```
  @Pointcut(“@within(com.amazonaws.xray.spring.aop.XRayEnabled) && bean(*Controller)”)
  ```

프로젝트에서 Spring 데이터 JPA를 사용하는 경우 BaseAbstractXRayInterceptor 대신 AbstractXRayInterceptor에서 확장하는 것을 고려하세요.

예제

다음 코드는 추상 클래스 BaseAbstractXRayInterceptor를 확장합니다.

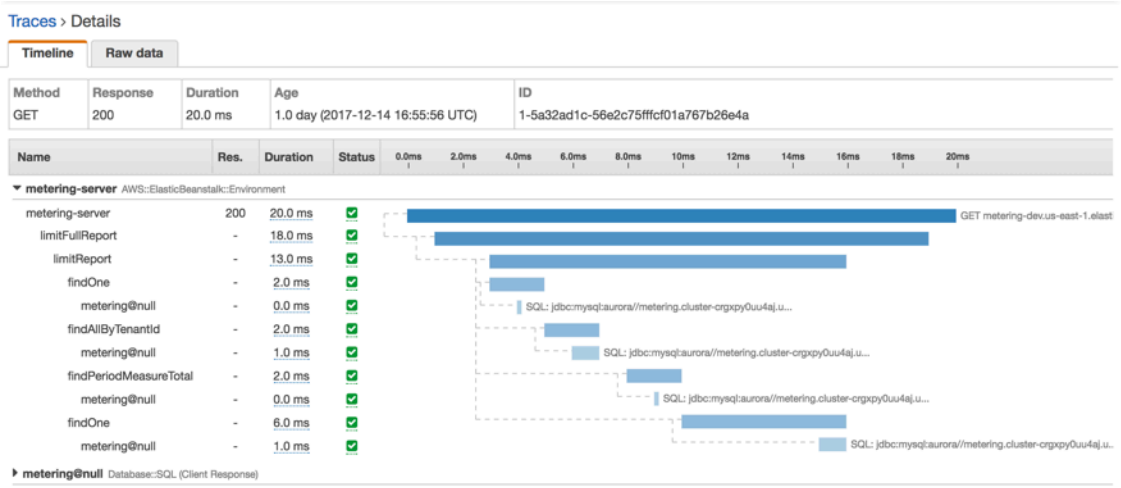
```
@Aspect
@Component
public class XRayInspector extends BaseAbstractXRayInterceptor {
    @Override
```

```
    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() {}

}
```

다음 코드는 X-Ray에 의해 구성될 클래스입니다.

```
@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());
        }
    }
}
```

애플리케이션을 올바로 구성했다면 다음 콘솔 스크린샷과 같이 컨트롤러에서 서비스 호출에 이르기까지 애플리케이션의 전체 호출 스택이 보일 것입니다.


Traces › Details
Timeline
Raw data
Method
Response
Duration
Age
ID
GET
200
20.0 ms
1.0 day (2017-12-14 16:55:56 UTC)
1-5a32ad1c-56e2c75fffcf01a767b26e4a
Name
Res.
Duration
Status
0.0ms
2.0ms
4.0ms
6.0ms
8.0ms
10ms
12ms
14ms
16ms
18ms
20ms
metering-server AWS::ElasticBeanstalk::Environment
metering-server 200 20.0 ms
limitFullReport - 18.0 ms
limitReport - 13.0 ms
findOne - 2.0 ms
metering@null - 0.0 ms
findAllByTenantId - 2.0 ms
metering@null - 1.0 ms
findPeriodMeasureTotal - 2.0 ms
metering@null - 0.0 ms
findOne - 6.0 ms
metering@null - 1.0 ms
GET metering-dev.us-east-1.elast
SQL: jdbc:mysql:aurora//metering.cluster-crgxpy0uu4aj.u...
SQL: jdbc:mysql:aurora//metering.cluster-crgxpy0uu4aj.u...
SQL: jdbc:mysql:aurora//metering.cluster-crgxpy0uu4aj.u...
SQL: jdbc:mysql:aurora//metering.cluster-crgxpy0uu4aj.u...
metering@null Database::SQL (Client Response)

Node.js 작업

Node.js 애플리케이션을 계측하여 트레이스를 X-Ray로 전송하는 방법에는 두 가지가 있습니다.

- AWS Distro for OpenTelemetry JavaScript - AWS Distro for OpenTelemetry Collector를 통해 상관 관계가 있는 지표 및 트레이스를 Amazon CloudWatch AWS X-Ray및 Amazon OpenSearch Service를 포함한 여러 AWS 모니터링 솔루션으로 전송하기 위한 오픈 소스 라이브러리 세트를 제공하는 AWS 배포입니다.
- AWS X-Ray Node.js용 SDK - X-Ray 데몬을 통해 트레이스를 생성하고 X-Ray로 전송하기 위한 라이브러리 세트입니다.

자세한 내용은 AWS Distro for OpenTelemetry와 X-Ray SDKs 중에서 선택 단원을 참조하십시오.

AWS Distro for OpenTelemetry JavaScript

AWS Distro for OpenTelemetry(ADOT) JavaScript를 사용하면 애플리케이션을 한 번 계측하고 상관관계가 있는 지표 및 추적을 Amazon CloudWatch AWS X-Ray, Amazon OpenSearch Service를 포함한 여러 AWS 모니터링 솔루션으로 전송할 수 있습니다. AWS Distro for OpenTelemetry와 함께 X-Ray를 사용하려면 X-Ray에서 사용할 수 있는 OpenTelemetry SDK와 X-Ray에서 사용할 수 있는 AWS Distro for OpenTelemetry Collector라는 두 가지 구성 요소가 필요합니다.

시작하려면 OpenTelemetry JavaScript용AWS 배포판 설명서를 참조하십시오.

> **Note**
>
> ADOT JavaScript는 모든 서버 측 Node.js 애플리케이션에서 지원됩니다. ADOT JavaScript는 브라우저 클라이언트에서 X-Ray로 데이터를 내보낼 수 없습니다.

AWS X-Ray 및 기타에서 AWS Distro for OpenTelemetry를 사용하는 방법에 대한 자세한 내용은 AWS Distro for OpenTelemetry 또는 AWS Distro for OpenTelemetry 설명서를 AWS 서비스참조하세요.

언어 지원 및 사용법에 대한 자세한 내용은 Github의AWS 관찰성을 참조하세요.

AWS Node.js용 X-Ray SDK

Node.js용 X-Ray SDK는 트레이스 데이터를 생성하고 X-Ray 대몬(daemon)에 보내기 위한 클래스 및 메서드를 제공하는 Express 웹 애플리케이션 및 Node.js Lambda 함수용 라이브러리입니다. 트레이스 데이터에는 애플리케이션에서 제공하는 수신 HTTP 요청과 애플리케이션이 AWS SDK 또는 HTTP 클라이언트를 사용하여 다운스트림 서비스에 수행하는 호출에 대한 정보가 포함됩니다.

> **Note**
>
> Node.js용 X-Ray SDK는 Node.js 버전 14.x 이상에서 지원되는 오픈 소스 프로젝트입니다. 프로젝트를 따르고 GitHub(github.com/aws/aws-xray-sdk-node)에서 문제 및 풀 요청을 제출할 수 있습니다.

Express를 사용한다면, 애플리케이션 서버에서 SDK를 미들웨어로 추가하여 수신 요청 트레이스를 시작합니다. 미들웨어는 각 트레이스 요청에 대한 세그먼트를 생성하고, 응답이 전송되면 세그먼트를 완료합니다. 세그먼트가 열려 있는 동안에는 SDK 클라이언트의 메서드를 이용해 정보를 세그먼트에 추가하고 하위 세그먼트를 만들어 다운스트림 호출을 트레이스할 수 있습니다. 또한 SDK는 세그먼트가 열려 있는 동안 애플리케이션에서 발생하는 예외를 자동으로 기록합니다.

계측되는 애플리케이션 또는 서비스에 의해 호출되는 Lambda 함수의 경우, Lambda는 추적 헤더를 읽고 샘플링된 요청을 자동으로 추적합니다. 그 밖의 함수의 경우 수신 요청을 샘플링 및 추적하도록 Lambda를 구성합니다. 어느 경우든, Lambda는 세그먼트를 생성하여 X-Ray SDK에 제공합니다.

> **Note**
>
> Lambda의 X-Ray SDK는 선택 사항입니다. 이를 함수에 사용하지 않는 경우 여전히 서비스 맵에 Lambda 서비스용 노드와 각 Lambda 함수용 노트 하나가 포함됩니다. SDK를 추가하면 함수 코드를 계측하여 Lambda에 의해 기록되는 함수 세그먼트에 하위 세그먼트를 추가할 수 있습니다. 자세한 내용은 AWS Lambda 그리고 AWS X-Ray 섹션을 참조하세요.

다음으로 Node.js용 X-Ray SDK를 사용하여 Node.js 클라이언트에서 JavaScript용 AWS SDK를 계측합니다. 계측된 클라이언트를 사용하여 다운스트림 AWS 서비스 또는 리소스를 호출할 때마다 SDK는 하위 세그먼트에 호출에 대한 정보를 기록합니다. 서비스 내에서 액세스하는 AWS 서비스 리소스는 트레이스 맵에 다운스트림 노드로 표시되므로 개별 연결에서 오류 및 제한 문제를 식별하는 데 도움이 됩니다.

또한 Node.js용 X-Ray SDK는 HTTP 웹 API 및 SQL 쿼리에 대한 다운스트림 호출의 계측도 제공합니다. HTTP 클라이언트를 SDK의 캡처 메서드에 래핑해 발신 HTTP 호출에 대한 정보를 기록합니다. SQL 클라이언트의 경우 데이터베이스 유형에 맞는 캡처 메서드를 사용하십시오.

미들웨어는 샘플링 규칙을 수신 요청에 적용해 트레이스할 요청을 결정합니다. Node.js용 X-Ray SDK를 구성하여 샘플링 동작을 조정하거나 애플리케이션이 실행되는 AWS 컴퓨팅 리소스에 대한 정보를 기록할 수 있습니다.

요청에 대한 추가 정보와 애플리케이션이 주석 및 메타데이터에서 하는 작업을 기록합니다. 주석은 필터 표현식과 함께 사용할 수 있도록 인덱싱된 단순한 키 값 쌍이기 때문에, 특정 데이터를 포함한 트레이스를 검색할 수 있습니다. 메타데이터 항목은 제한이 적으며 JSON으로 직렬화할 수 있는 모든 객체와 어레이를 기록할 수 있습니다.

> **주석 및 메타데이터**
>
> 주석 및 메타데이터는 X SDK를 사용하여 세그먼트에 추가하는 임의의 텍스트입니다. 주석은 필터 표현식에서 사용하기 위해 인덱싱됩니다. 메타데이터는 인덱싱되지 않지만 X-Ray 콘솔 또는 API를 사용하여 원시 세그먼트에서 볼 수 있습니다. X-Ray에 대한 읽기 액세스가 부여된 사용자는 누구나 이 데이터를 볼 수 있습니다.

코드에 구성된 클라이언트가 많이 있다면, 구성된 클라이언트로 만든 각 직접 호출의 하위 세그먼트를 대량으로 보관하는 요청 세그먼트 하나를 만들 수 있습니다. 사용자 지정 하위 세그먼트의 클라이언트 호출을 래핑해 하위 세그먼트를 조직하고 그룹화할 수 있습니다. 전체 함수나 특정 코드 부분에 대한 사용자 지정 하위 세그먼트를 만들고, 상위 세그먼트에 모든 것을 적는 대신 하위 세그먼트에 메타데이터와 주석을 기록할 수 있습니다.

SDK의 클래스 및 메서드에 대한 레퍼런스 문서는 Node.js용AWS X-Ray X-Ray SDK API Reference를 참조하십시오.

요구 사항

Node.js용 X-Ray SDK는 Node.js 및 다음 라이브러리가 필요합니다.

- `atomic-batcher` – 1.0.2
- `cls-hooked` – 4.2.2
- `pkginfo` – 0.4.0
- `semver` – 5.3.0

SDK는 사용자가 NPM으로 설치할 때 이 라이브러리를 가져옵니다.

AWS SDK 클라이언트를 추적하려면 Node.js용 X-Ray SDK에 Node.js의 JavaScript용 AWS SDK의 최소 버전이 필요합니다.

- `aws-sdk` – 2.7.15

종속성 관리

Node.js용 X-Ray SDK는 NPM에서 사용할 수 있습니다.

- 패키지 – `aws-xray-sdk`

로컬 개발의 경우에는 NPM으로 SDK를 프로젝트 디렉터리에 설치합니다.

```
~/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
```

`--save` 옵션을 사용해 SDK를 애플리케이션의 `package.json`에 종속성으로 저장합니다.

```
~/nodejs-xray$ npm install aws-xray-sdk --save
aws-xray-sdk@3.3.3
```

애플리케이션에 X-Ray SDK의 종속성과 충돌하는 버전이 있는 종속성이 있는 경우, 호환성을 보장하기 위해 두 버전이 모두 설치됩니다. 자세한 내용은 종속성 해결에 대한 공식 NPM 문서를 참조하십시오.

Node.js 샘플

AWS X-Ray SDK for Node.js를 사용하여 요청이 Node.js 애플리케이션을 통해 이동할 때 end-to-end 볼 수 있습니다.

- GitHub의 Node.js 샘플 애플리케이션.

Node.js용 X-Ray SDK 구성

애플리케이션이 실행되는 서비스에 대한 정보를 포함하거나, 기본 샘플링 동작을 수정하거나 요청에 적용되는 샘플링 규칙을 특정 경로에 추가하도록 플러그인을 사용하여 Node.js용 X-Ray SDK를 구성할 수 있습니다.

Sections

- 서비스 플러그인
- 샘플링 규칙
- 로깅
- X-Ray 대몬(daemon) 주소
- 환경 변수

서비스 플러그인

`plugins`을 사용하여 애플리케이션을 호스팅하는 서비스에 대한 정보를 기록할 수 있습니다.

플러그인

- Amazon EC2 — `EC2Plugin`은 인스턴스 ID, 가용 영역 및 CloudWatch Logs 그룹을 추가합니다.
- Elastic Beanstalk – `ElasticBeanstalkPlugin`이 환경 이름, 버전 레이블 및 배포 ID를 추가합니다.
- Amazon ECS — `ECSPlugin`이 컨테이너 ID를 추가합니다.

플러그인을 사용하려면 `config` 메서드를 사용하여 Node.js용 X-Ray SDK 클라이언트를 구성합니다.

Example app.js - 플러그인

```
var AWSXRay = require('aws-xray-sdk');
AWSXRay.config([AWSXRay.plugins.EC2Plugin,AWSXRay.plugins.ElasticBeanstalkPlugin]);
```

SDK는 플러그인 설정을 사용하여 세그먼트에 `origin` 필드를 설정하기도 합니다. 이는 애플리케이션을 실행하는 AWS 리소스 유형을 나타냅니다. 여러 플러그인을 사용하는 경우 SDK는 ElasticBeanstalk> EKS> ECS> EC2 순서로 확인하여 오리진을 결정합니다.

샘플링 규칙

SDK는 X-Ray 콘솔에서 정의하는 샘플링 규칙을 사용하여 기록할 요청을 결정합니다. 기본 규칙은 매초 첫 번째 요청을 추적하고, 모든 서비스에서 추가 요청의 5%를 X-Ray로 추적 전송합니다. X-Ray 콘솔에서 추가 규칙을 생성하여 각 애플리케이션에 대해 기록되는 데이터의 양을 사용자 지정합니다.

SDK는 사용자 지정 규칙을 정의된 순서대로 적용합니다. 요청이 여러 사용자 지정 규칙과 일치하는 경우 SDK는 첫 번째 규칙만 적용합니다.

> **Note**
>
> SDK가 샘플링 규칙을 가져오기 위해 X-Ray에 연결할 수 없는 경우, 매초 첫 번째 요청과 호스트당 추가 요청의 5%에 대한 기본 로컬 규칙으로 되돌아갑니다. 호스트가 샘플링 API를 직접 호출할 수 있는 권한이 없거나, X-Ray 대몬(daemon)에 연결할 수 없을 경우 이러한 상황이 발생할 수 있고 이 대몬(daemon)은 SDK에서 수행한 API 직접 호출에 대한 TCP 프록시 역할을 합니다.

JSON 문서에서 샘플링 규칙을 불러오도록 SDK를 구성할 수도 있습니다. SDK는 X-Ray 샘플링을 사용할 수 없을 경우 로컬 규칙을 백업으로 사용하거나, 로컬 규칙을 전용으로 사용할 수 있습니다.

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
  }
}
```

이 예에서는 하나의 사용자 지정 규칙과 기본 규칙을 정의합니다. 사용자 지정 규칙은 최소 추적 요청 수 없이 5% 샘플링 비율을 `/api/move/` 아래 경로에 적용합니다. 기본 규칙은 매초 최초 요청과 추가 요청의 10%를 추적합니다.

로컬로 규칙의 정의할 때의 단점은 고정 대상이 X-Ray 서비스를 통해 관리되는 대신, 레코더의 각 인스턴스별로 독립적으로 적용된다는 것입니다. 호스트를 많이 배포할수록 고정 속도가 크게 증대하기 때문에 기록되는 데이터의 양을 제어하기가 어려워집니다.

에서는 샘플링 속도를 수정할 수 AWS Lambda없습니다. 구성된 서비스가 함수를 호출하는 경우 해당 서비스에서 샘플링한 요청을 생성한 호출이 Lambda에 의해 기록됩니다. 활성 추적이 활성화되고 트레이스 헤더가 없는 경우 Lambda에서 샘플링 결정을 내립니다.

백업 규칙을 구성하려면 `setSamplingRules`를 사용하여 파일에서 샘플링 규칙을 로드하도록 Node.js용 X-Ray SDK에 지정합니다.

Example app.js - 파일의 샘플링 규칙

```
var AWSXRay = require('aws-xray-sdk');
AWSXRay.middleware.setSamplingRules('sampling-rules.json');
```

또한 코드에서 규칙을 정의하여 객체로 `setSamplingRules`에 전달할 수도 있습니다.

Example app.js - 객체의 샘플링 규칙

```
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);
```

로컬 규칙만 사용하려면 disableCentralizedSampling를 호출합니다.

```
AWSXRay.middleware.disableCentralizedSampling()
```

로깅

SDK의 출력을 기록하려면 AWSXRay.setLogger(logger)를 호출합니다. 여기서 logger는 표준 로깅 메서드(warn, info 등)를 제공하는 객체입니다.

기본적으로 SDK는 콘솔 객체의 표준 메서드를 사용하여 콘솔에 오류 메시지를 기록합니다. 내장 로거의 로그 수준은 AWS_XRAY_DEBUG_MODE 또는 AWS_XRAY_LOG_LEVEL 환경 변수를 사용하여 설정할 수 있습니다. 유효한 로그 수준 값 목록은 환경 변수를 참조하십시오.

로그에 다른 형식이나 대상을 제공하려는 경우 아래와 같이 로거 인터페이스의 자체 구현을 SDK에 제공할 수 있습니다. 이 인터페이스를 구현하는 모든 객체를 사용할 수 있습니다. 즉, Winston과 같은 다양한 로깅 라이브러리를 사용하여 SDK에 직접 전달할 수 있습니다.

Example app.js – 로깅

```
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]);
```

다른 구성 메서드를 실행하기 전에 setLogger를 호출하여 해당 작업의 출력을 캡처하도록 합니다.

X-Ray 대몬(daemon) 주소

X-Ray 대몬(daemon)이 127.0.0.1:2000 이외의 다른 포트 또는 호스트를 수신 대기하는 경우 추적 데이터를 다른 주소로 전송하도록 Node.js용 X-Ray SDK를 구성할 수 있습니다.

```
AWSXRay.setDaemonAddress('host:port');
```

호스트를 이름 또는 IPv4 주소로 지정할 수 있습니다.

Example app.js - 데몬 주소

```
var AWSXRay = require('aws-xray-sdk');
AWSXRay.setDaemonAddress('daemonhost:8082');
```

다른 포트에서 TCP와 UDP를 수신 대기하도록 데몬을 구성한 경우 데몬 주소 설정에서 둘 다 지정할 수 있습니다.

Example app.js - 개별 포트의 데몬 주소

```
var AWSXRay = require('aws-xray-sdk');
AWSXRay.setDaemonAddress('tcp:daemonhost:8082 udp:daemonhost:8083');
```

AWS_XRAY_DAEMON_ADDRESS 환경 변수를 사용하여 데몬 주소를 설정할 수도 있습니다.

환경 변수

환경 변수를 사용하여 Node.js용 X-Ray SDK를 구성할 수 있습니다. SDK는 다음 변수를 지원합니다.

- AWS_XRAY_CONTEXT_MISSING – 열려 있는 세그먼트가 없는 경우 구성된 구성된 코드가 데이터를 기록하려고 할 때 발생하는 예외를 방지하려면 RUNTIME_ERROR로 설정합니다.

  유효한 값

  - RUNTIME_ERROR— 런타임 예외가 발생합니다.
  - LOG_ERROR – 오류를 기록하고 계속합니다 (기본값).
  - IGNORE_ERROR— 오류를 무시하고 계속합니다.

  열린 요청이 없을 때 실행되는 시작 코드 또는 새 스레드를 생성하는 코드에서 계측된 클라이언트를 사용하려고 하는 경우 누락된 세그먼트 또는 하위 세그먼트와 관련된 오류가 발생할 수 있습니다.
- AWS_XRAY_DAEMON_ADDRESS – X-Ray 대몬(daemon) 리스너의 호스트와 포트를 설정합니다. 기본적으로 SDK는 추적 데이터(UDP)와 샘플링(TCP) 모두에 127.0.0.1:2000을 사용합니다. 다른 포트에서 수신 대기하도록 대몬(daemon)을 구성한 경우 또는 다른 호스트에서 실행 중인 경우 이 변수를 사용합니다.

  형식

  - 동일한 포트 – *address*:*port*

  - 다른 포트 – `tcp:address:port udp:address:port`
- `AWS_XRAY_DEBUG_MODE` – debug 수준에서 로그를 콘솔에 출력하도록 SDK를 구성하려면 `TRUE`로 설정합니다.
- `AWS_XRAY_LOG_LEVEL` – 기본 로거의 로그 수준을 설정합니다. 유효한 값은 `debug`, `info`, `warn`, `error` 및 `silent`입니다. AWS_XRAY_DEBUG_MODE가 `TRUE`로 설정된 경우 이 값은 무시됩니다.
- `AWS_XRAY_TRACING_NAME` – SDK가 세그먼트에 사용할 서비스 이름을 설정합니다. Express 미들웨어에 설정한 세그먼트 이름을 재정의합니다.

Node.js용 X-Ray SDK로 수신 요청 추적하기

Node.js용 X-Ray SDK를 사용하여 Express 및 Restify 애플리케이션이 Amazon EC2 AWS Elastic Beanstalk또는 Amazon ECS의 EC2 인스턴스에서 제공하는 수신 HTTP 요청을 추적할 수 있습니다.

Node.js용 X-Ray SDK는 Express 및 Restify 프레임워크를 사용하는 애플리케이션을 위한 미들웨어를 제공합니다. 애플리케이션에 X-Ray 미들웨어를 추가하면 Node.js용 X-Ray SDK가 샘플링된 각 요청에 대해 세그먼트를 생성합니다. 이 세그먼트에는 HTTP 요청의 시간, 메서드 및 배치가 포함됩니다. 추가로 구성하면 이 세그먼트의 하위 세그먼트가 생성됩니다.

> **Note**
>
> AWS Lambda 함수의 경우 Lambda는 샘플링된 각 요청에 대해 세그먼트를 생성합니다. 자세한 내용은 AWS Lambda 그리고 AWS X-Ray 섹션을 참조하세요.

각 세그먼트에는 서비스 맵 안에서 애플리케이션을 식별하는 이름이 있습니다. 이 세그먼트의 이름이 정적으로 지정되도록 하거나, 수신 요청의 호스트 헤더를 기반으로 SDK가 동적으로 이름을 지정하도록 구성할 수 있습니다. 동적 이름 지정을 사용하면 요청의 도메인 이름에 따라 그룹을 추적하고 이름이 예상 패턴과 일치하지 않을 경우(예: 호스트 헤더가 위조된 경우) 기본 이름을 적용할 수 있습니다.

> **전달된 요청**
>
> 로드 밸런서 또는 기타 중개자가 애플리케이션으로 요청을 전달하는 경우, X-Ray는 IP 패킷 내 소스 IP가 아니라 요청의 `X-Forwarded-For` 헤더로부터 클라이언트 IP를 가져옵니다. 전달된 요청에 대해 기록된 클라이언트 IP는 위조될 수 있으므로 신뢰하면 안 됩니다.

요청이 전달되면 SDK는 세그먼트에 추가 필드를 설정하여 이를 나타냅니다. 세그먼트에 `x_forwarded_for`로 설정된 `true` 필드가 포함된 경우 클라이언트 IP는 HTTP 요청의 `X-Forwarded-For` 헤더로부터 가져옵니다.

메시지 핸들러는 다음 정보를 포함하는 `http` 블록을 이용해 각 수신 요청에 대한 세그먼트를 생성합니다.

- HTTP 메서드 – GET, POST, PUT, DELETE 등.
- 클라이언트 주소 – 요청을 전송한 클라이언트의 IP 주소.
- 응답 코드 – 완료된 요청의 HTTP 응답 코드.
- 시간 – 시작 시간(요청 수신) 및 종료 시간(응답 전송).
- 유저 에이전트 — 요청에서 가져온 `user-agent`입니다.
- 콘텐츠 길이 — 응답의 `content-length`입니다.

Sections

Express를 사용하여 수신 요청 추적

Express 미들웨어를 사용하려면 SDK 클라이언트를 시작하고 `express.openSegment` 함수가 반환하는 미들웨어를 사용하여 경로를 정의합니다.

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());
```

경로를 정의한 후, `express.closeSegment`의 출력을 사용하여 Node.js용 X-Ray SDK가 반환한 오류를 모두 처리합니다.

Restify를 사용하여 수신 요청 추적

Restify 미들웨어를 사용하려면 SDK 클라이언트를 시작하고 `enable`을 실행합니다. Restify 서버 및 세그먼트 이름을 전달합니다.

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');
});
```

세그먼트 이름 지정 전략 구성

AWS X-Ray 는 서비스 이름을 사용하여 애플리케이션을 식별하고 애플리케이션이 사용하는 다른 애플리케이션, 데이터베이스, 외부 APIs 및 AWS 리소스와 구분합니다. X-Ray SDK는 수신 요청에 대한 세그먼트를 생성할 때 해당 세그먼트의 이름 필드에 애플리케이션의 서비스 이름을 기록합니다.

X-Ray SDK는 HTTP 요청 헤더의 호스트 이름 뒤에 세그먼트를 지정할 수 있습니다. 그러나 이 헤더가 위조되면 서비스 맵에 예기치 않은 노드가 발생할 수 있습니다. 위조된 호스트 헤더가 포함된 요청으로 인해 SDK가 잘못된 세그먼트 이름을 지정하는 현상을 방지하려면 들어오는 요청의 기본 이름을 지정해야 합니다.

애플리케이션이 여러 도메인의 요청을 처리하는 경우, 동적 이름 지정 전략을 사용하여 이를 세그먼트 이름에 반영하도록 SDK를 구성할 수 있습니다. 동적 이름 지정 전략을 사용하면 SDK가 예상 패턴과 일치하는 요청에 호스트 이름을 사용하고, 그렇지 않은 요청에 기본 이름을 적용할 수 있습니다.

예를 들어, 하나의 애플리케이션이 세 개의 하위 도메인 (`www.example.com`, `api.example.com`, `static.example.com`) 에 요청을 전송할 수 있습니다. `*.example.com` 패턴으로 동적 이름 지정 전략을 사용하여 각 하위도메인의 세그먼트를 서로 다른 이름으로 표시하면 서비스 맵에 서비스 노드가 세 개 생깁니다. 이 패턴에 맞지 않는 호스트 이름의 요청이 애플리케이션에 수신되면, 사용자가 지정한 대체 이름의 네 번째 노드가 서비스 맵에 표시됩니다.

모든 요청 세그먼트에 같은 이름을 사용하려면, 이전 단원에 나온 것처럼 미들웨어를 초기화할 때 애플리케이션 이름을 지정하십시오.

> **Note**
>
> 코드에 정의한 기본 서비스 이름을 AWS_XRAY_TRACING_NAME 환경 변수를 사용하여 재정의할 수 있습니다.

동적 이름 지정 전략은 호스트 이름이 일치해야 하는 패턴 및 HTTP 요청의 호스트 이름이 패턴과 일치하지 않는 경우 사용할 기본 이름을 정의합니다. 동적으로 세그먼트의 이름을 지정하려면 AWSXRay.middleware.enableDynamicNaming을 사용합니다.

Example app.js – 동적 세그먼트 이름

요청의 호스트 이름이 *.example.com 패턴과 일치하는 경우 호스트 이름을 사용합니다. 그렇지 않은 경우 MyApp을 사용합니다.

```
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());
```

Node.js용 X-Ray AWS SDK를 사용하여 SDK 호출 추적

애플리케이션에서를 호출 AWS 서비스 하여 데이터를 저장하거나, 대기열에 쓰거나, 알림을 보내면 Node.js용 X-Ray SDK는 하위 세그먼트에서 다운스트림으로 호출을 추적합니다. 트레이스 AWS 서비스 및 해당 서비스 내에서 액세스하는 리소스(예: Amazon S3 버킷 또는 Amazon SQS 대기열)는 X-Ray 콘솔의 트레이스 맵에 다운스트림 노드로 표시됩니다.

AWS SDK for JavaScript V2 또는 AWS SDK for JavaScript V3를 통해 생성한 AWS SDK 클라이언트를 계측합니다. 각 AWS SDK 버전은 AWS SDK 클라이언트를 계측하기 위한 다양한 방법을 제공합니다.

> **Note**
>
> 현재 Node.js용 AWS X-Ray SDK는 AWS SDK for JavaScript V2 클라이언트를 계측할 때와 비교하여 V3 클라이언트를 계측할 때 더 적은 세그먼트 정보를 반환합니다. V2 예를 들어, DynamoDB에 대한 직접 호출을 나타내는 하위 세그먼트는 테이블 이름을 반환하지 않습니다. 트레이스에이 세그먼트 정보가 필요한 경우 AWS SDK for JavaScript V2 사용을 고려하세요.

AWS SDK for JavaScript V2

에 대한 호출에서 `aws-sdk` 필수 문을 래핑하여 모든 AWS SDK V2 클라이언트를 계측할 수 있습니다`AWSXRay.captureAWS`.

Example app.js - AWS SDK 구성

```
const AWS = AWSXRay.captureAWS(require('aws-sdk'));
```

개별 클라이언트를 계측하려면에 대한 호출로 AWS SDK 클라이언트를 래핑합니다`AWSXRay.captureAWSClient`. 예를 들어, `AmazonDynamoDB` 클라이언트를 구성하려면

Example app.js - DynamoDB 클라이언트 계측

```
    const AWSXRay = require('aws-xray-sdk');
...
    const ddb = AWSXRay.captureAWSClient(new AWS.DynamoDB());
```

> **Warning**
>
> `captureAWS` 및 `captureAWSClient`를 함께 사용하지 마십시오. 이렇게 하면 하위 세그먼트가 중복됩니다.

TypeScript와 ECMAScript 모듈(ESM)을 사용하여 JavaScript 코드를 로드하려는 경우 다음 예제를 사용하여 라이브러리를 가져옵니다.

Example app.js - AWS SDK 계측

```
import * as AWS from 'aws-sdk';
import * as AWSXRay from 'aws-xray-sdk';
```

ESM을 사용하여 모든 AWS 클라이언트를 계측하려면 다음 코드를 사용합니다.

Example app.js - AWS SDK 계측

```
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();
```

모든 서비스의 경우, X-Ray 콘솔에서 호출된 API의 이름을 볼 수 있습니다. 서비스 하위 집합에 대해서는 X-Ray SDK가 세그먼트에 정보를 추가하여 서비스 맵에서 추가 세분화를 제공합니다.

예를 들어 계측된 DynamoDB 클라이언트에서 직접 호출을 생성하는 경우 SDK가 특정 테이블을 대상으로 한 직접 호출에 대해 테이블 이름을 세그먼트에 추가합니다. 콘솔에서, 각 테이블이 개별 노드로 서비스 맵에 표시되고, 특정 테이블을 대상으로 하지 않은 직접 호출에 대해 일반 DynamoDB 노드가 표시됩니다.

Example 항목을 저장하기 위한 DynamoDB 직접 호출에 대한 하위 세그먼트

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

명명된 리소스에 액세스할 때 다음 서비스를 호출할 경우 서비스 맵에 추가 노드가 생성됩니다. 특정 리소스를 대상으로 하지 않는 경우 서비스를 직접 호출하면 해당 서비스에 대한 일반 노드가 생성됩니다.

- Amazon DynamoDB – 테이블 이름
- Amazon Simple Storage Service –버킷 및 키 이름
- Amazon Simple Queue Service – 대기열 이름

AWS SDK for JavaScript V3

AWS SDK for JavaScript V3는 모듈식이므로 코드는 필요한 모듈만 로드합니다. 따라서 V3는 `captureAWS` 메서드를 지원하지 않으므로 모든 AWS SDK 클라이언트를 계측할 수 없습니다.

TypeScript와 ECMAScript 모듈(ESM)을 사용하여 JavaScript 코드를 로드하려는 경우 다음 예제를 사용하여 라이브러리를 가져올 수 있습니다.

```
import * as AWS from 'aws-sdk';
import * as AWSXRay from 'aws-xray-sdk';
```

`AWSXRay.captureAWSv3Client` 메서드를 사용하여 각 AWS SDK 클라이언트를 계측합니다. 예를 들어, `AmazonDynamoDB` 클라이언트를 구성하려면

Example app.js - Javascript V3용 SDK를 사용한 DynamoDB 클라이언트 계측

```
    const AWSXRay = require('aws-xray-sdk');
    const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
...
    const ddb = AWSXRay.captureAWSv3Client(new DynamoDBClient({ region:
 "region" }));
```

AWS SDK for JavaScript V3를 사용하는 경우 테이블 이름, 버킷 및 키 이름 또는 대기열 이름과 같은 메타데이터는 현재 반환되지 않으므로 V AWS SDK for JavaScript V2를 사용하여 AWS SDK 클라이언트를 계측할 때와 마찬가지로 트레이스 맵에는 이름이 지정된 각 리소스에 대한 개별 노드가 포함되지 않습니다.

Example AWS SDK for JavaScript V3 사용 시 DynamoDB에 대한 호출을 위한 하위 세그먼트를 사용하여 항목 저장

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

Node.js용 X-Ray SDK를 사용하여 다운스트림 HTTP 웹 서비스에 대한 호출 추적하기

애플리케이션이 마이크로서비스 또는 공개 HTTP API를 호출할 때 Node.js용 X-Ray SDK 클라이언트를 사용하여 해당 호출을 계측하고 API를 다운스트림 서비스로 서비스 그래프에 추가할 수 있습니다.

`http` 또는 `https` 클라이언트를 Node.js용 X-Ray SDK의 `captureHTTPs` 메서드로 전달하여 발신 호출을 트레이스합니다.

> **Note**
>
> Axios 또는 Superagent 같은 타사 HTTP 요청 라이브러리를 사용하는 호출은 `captureHTTPsGlobal()` API를 통해 지원되며 네이티브 `http` 모듈을 사용할 때 추적됩니다.

Example app.js – HTTP 클라이언트

```
var AWSXRay = require('aws-xray-sdk');
var http = AWSXRay.captureHTTPs(require('http'));
```

모든 HTTP 클라이언트에서 추적을 활성화하려면 `http`를 로드하기 전에 `captureHTTPsGlobal`을 호출합니다.

Example app.js – HTTP 클라이언트(전역)

```
var AWSXRay = require('aws-xray-sdk');
```

```
AWSXRay.captureHTTPsGlobal(require('http'));
var http = require('http');
```

다운스트림 웹 API에 대한 직접 호출을 계측할 때 Node.js용 X-Ray SDK가 HTTP 요청 및 응답에 대한 정보가 포함된 하위 세그먼트를 기록합니다. X-Ray는 하위 세그먼트를 사용하여 원격 API에 대해 추정된 세그먼트를 생성합니다.

Example 다운스트림 HTTP 호출에 대한 하위 세그먼트

```
{
  "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 다운스트림 HTTP 호출에 대한 추정된 세그먼트

```
{
  "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
}
```

Node.js용 X-Ray SDK로 SQL 쿼리 추적하기

SQL 클라이언트를 대응하는 Node.js용 X-Ray SDK 클라이언트 메서드로 감싸 SQL 데이터베이스 쿼리를 계측합니다.

- 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);
```

구성된 클라이언트를 이용해 SQL 쿼리를 생성한다면, Node.js용 X-Ray SDK는 연결과 쿼리에 대한 정보를 하위 세그먼트에 기록합니다.

SQL 하위 세그먼트에 추가 데이터 포함

허용 목록에 있는 SQL 필드에 매핑된다면 SQL 쿼리에 대해 생성된 하위 세그먼트에 정보를 더 추가할 수 있습니다. 예를 들어 하위 세그먼트에 삭제된 SQL 쿼리 문자열을 기록하려는 경우 하위 세그먼트의 SQL 객체에 직접 추가할 수 있습니다.

Example 하위 세그먼트에 SQL 지정

```
    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;
}
```

허용 목록에 있는 SQL 필드의 전체 목록은 AWS X-Ray 개발자 안내서의 SQL 쿼리를 참조하십시오.

Node.js용 X-Ray SDK를 사용하여 사용자 지정 하위 세그먼트 생성하기

하위 세그먼트는 추적의 세그먼트를 확장하여 요청을 처리하기 위해 완료된 작업에 대한 세부 정보를 표시합니다. 계측되는 클라이언트에서 직접 호출할 때마다, X-Ray SDK는 하위 세그먼트 안에 생성된 정보를 기록합니다. 추가 하위 세그먼트를 생성하여 다른 하위 세그먼트를 그룹화하거나, 코드 섹션의 성능을 평가하거나, 주석 및 메타데이터를 기록할 수 있습니다.

사용자 지정 Express 하위 세그먼트

`captureAsyncFunc` 함수를 사용하여 다운스트림 서비스를 호출하는 기능에 대해 사용자 지정 하위 세그먼트를 생성할 수 있습니다.

Example app.js – 사용자 지정 하위 세그먼트 Express

```
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();
};
```

이 예제에서 애플리케이션은 sendRequest 함수를 호출하기 위해 send라는 사용자 지정 하위 세그먼트를 생성합니다. captureAsyncFunc는 콜백 함수가 생성하는 비동기 호출이 완료되면 해당 콜백 함수 내에서 닫아야 하는 하위 세그먼트를 전달합니다.

동기식 기능에 대해서는 captureFunc 함수를 사용할 수 있습니다. 이 함수는 함수 블록이 실행을 마치는 즉시 자동으로 하위 세그먼트를 닫습니다.

하위 세그먼트를 세그먼트 또는 다른 하위 세그먼트 내에서 생성할 경우 Node.js용 X-Ray SDK가 해당 하위 세그먼트에 대해 ID를 생성하고 시작 시간 및 종료 시간을 기록합니다.

Example 메타데이터가 포함된 하위 세그먼트

```
"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"
    }
```

```
  },
```

사용자 지정 Lambda 하위 세그먼트

SDK는 Lambda에서 실행 중임을 감지하면 자리표시자 facade 세그먼트를 자동으로 생성하도록 구성됩니다. X-Ray 트레이스 맵에 단일 `AWS::Lambda::Function` 노드를 생성할 기본 하위 세그먼트를 생성하려면 facade 세그먼트를 직접 호출하고 용도를 변경합니다. 추적 ID, 상위 ID 및 샘플링 결정을 공유할 때 새 ID를 사용하여 새 세그먼트를 수동으로 만들면 새 세그먼트를 보낼 수 있습니다.

Example app.js – 수동 사용자 지정 하위 세그먼트

```
const segment = AWSXRay.getSegment(); //returns the facade segment
const subsegment = segment.addNewSubsegment('subseg');
...
subsegment.close();
//the segment is closed by the SDK automatically
```

Node.js용 X-Ray SDK로 세그먼트에 주석 및 메타데이터 추가하기

주석 및 메타데이터와 함께 요청, 환경 또는 애플리케이션에 대한 추가 정보를 기록할 수 있습니다. X-Ray SDK에서 생성하는 세그먼트 또는 사용자가 생성하는 사용자 지정 하위 세그먼트에 주석 및 메타데이터를 추가할 수 있습니다.

주석은 문자열, 숫자 또는 부울 값과 결합한 키-값 페어입니다. 주석은 필터 표현식에서 사용하기 위해 인덱싱됩니다. 주석은 콘솔의 트레이스를 그룹화할 때 사용할 데이터를 기록하거나 `GetTraceSummaries` API를 직접 호출할 때 사용하세요.

메타데이터는 객체 및 목록을 포함한 모든 유형의 값을 가질 수 있는 키-값 페어지만, 필터 표현식에 사용할 수 있도록 인덱싱되지는 않습니다. 트레이스에 저장하고 싶지만 검색에는 사용하지 않을 추가 데이터는 메타데이터를 사용하여 기록하십시오.

세그먼트에는 주석과 메타데이터 외에 사용자 ID 문자열도 기록할 수 있습니다. 사용자 ID는 세그먼트의 별도 필드에 기록되면 검색용으로 인덱스되지 않습니다.

Sections

- Node.js용 X-Ray SDK로 주석 기록하기
- Node.js용 X-Ray SDK로 메타데이터 기록하기
- Node.js용 X-Ray SDK로 사용자 ID 기록하기

Node.js용 X-Ray SDK로 주석 기록하기

주석을 사용하여 검색용으로 인덱싱할 정보를 세그먼트나 하위 세그먼트에 기록하십시오.

주석 요구 사항

- 키 - X-Ray 주석의 키는 최대 500자의 영숫자를 포함할 수 있습니다. 점이나 마침표(.) 이외의 공백이나 기호를 사용할 수 없습니다.
- 값 - X-Ray 주석의 값은 최대 1,000자의 유니코드 문자를 포함할 수 있습니다.
- 주석 수 - 트레이스당 최대 50개의 주석을 사용할 수 있습니다.

주석 기록 방법

1. 현재 세그먼트나 하위 세그먼트의 참조를 가져옵니다.

   ```
   var AWSXRay = require('aws-xray-sdk');
   ...
   var document = AWSXRay.getSegment();
   ```

2. 문자열 키, 부울, 숫자 또는 문자열 값으로 addAnnotation을 직접 호출합니다.

   ```
   document.addAnnotation("mykey", "my value");
   ```

   다음 예에서는 점이 포함된 문자열 키와 부울, 숫자 또는 문자열 값을 사용하여 putAnnotation을 직접 호출하는 방법을 보여줍니다.

   ```
   document.putAnnotation("testkey.test", "my value");
   ```

SDK는 세그먼트 문서의 annotations 객체에 주석을 키-값 페어로 기록합니다. 같은 키로 addAnnotation을 두 번 직접 호출하면 같은 세그먼트나 하위 세그먼트에 기록했던 값을 덮어씁니다.

특정 값을 포함한 주석이 있는 트레이스를 찾으려면 annotation[*key*] 키워드를 필터 표현식에 사용하십시오.

Example app.js - 주석

```
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) {
...
```

Node.js용 X-Ray SDK로 메타데이터 기록하기

메타데이터를 이용해 검색용으로 인덱싱하지 않아도 되는 정보를 세그먼트나 하위 세그먼트에 기록하십시오. 메타데이터 값은 문자열, 숫자, 부울 또는 JSON 객체나 어레이에 직렬화할 수 있는 다른 모든 객체가 될 수 있습니다.

메타데이터 기록 방법

1. 현재 세그먼트나 하위 세그먼트의 참조를 가져옵니다.

   ```
   var AWSXRay = require('aws-xray-sdk');
   ...
   var document = AWSXRay.getSegment();
   ```

2. 문자열 키, 부울, 숫자, 문자열 또는 객체 값 및 문자열 네임스페이스로 `addMetadata`를 호출합니다.

   ```
   document.addMetadata("my key", "my value", "my namespace");
   ```

   or

   키와 값만 이용해 `addMetadata`를 직접 호출합니다.

```
document.addMetadata("my key", "my value");
```

네임스페이스를 지정하지 않으면, SDK는 `default`를 사용합니다. 같은 키로 `addMetadata`을 두 번 직접 호출하면 같은 세그먼트나 하위 세그먼트에 기록했던 값을 덮어씁니다.

Node.js용 X-Ray SDK로 사용자 ID 기록하기

사용자 ID를 요청 세그먼트에 기록하여 요청을 보낸 사용자를 식별합니다. Lambda 환경의 세그먼트는 변경할 수 없으므로이 작업은 AWS Lambda 함수와 호환되지 않습니다. `setUser` 호출은 하위 세그먼트가 아닌 세그먼트에만 적용할 수 있습니다.

사용자 ID 기록 방법

1. 현재 세그먼트나 하위 세그먼트의 참조를 가져옵니다.

   ```
   var AWSXRay = require('aws-xray-sdk');
   ...
   var document = AWSXRay.getSegment();
   ```

2. 요청을 보낸 사용자의 문자열 ID로 `setUser()`를 직접 호출합니다.

   ```
   var user = 'john123';

   AWSXRay.getSegment().setUser(user);
   ```

빠른 애플리케이션이 요청 처리를 시작하는 즉시 사용자 ID를 기록하기 위해 `setUser`를 호출할 수 있습니다. 사용자 ID 설정을 위해서만 세그먼트를 사용한다면 호출을 1줄로 연결할 수 있습니다.

Example app.js - 사용자 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) {
...
```

사용자 ID의 트레이스를 찾으려면, `user` 키워드를 필터 표현식에 적용하십시오.

Python 작업

Python 애플리케이션을 계측하여 트레이스를 X-Ray로 전송하는 방법에는 두 가지가 있습니다.

- AWS Distro for OpenTelemetry Python - AWS Distro for OpenTelemetry Collector를 통해 상관관계가 있는 지표 및 트레이스를 Amazon CloudWatch AWS X-Ray및 Amazon OpenSearch Service를 포함한 여러 AWS 모니터링 솔루션으로 전송하기 위한 오픈 소스 라이브러리 세트를 제공하는 AWS 배포입니다.
- AWS X-Ray SDK for Python - X-Ray 데몬을 통해 트레이스를 생성하고 X-Ray로 전송하기 위한 라이브러리 세트입니다.

자세한 내용은 AWS Distro for OpenTelemetry와 X-Ray SDKs 중에서 선택 단원을 참조하십시오.

AWS Distro for OpenTelemetry Python

AWS Distro for OpenTelemetry(ADOT) Python을 사용하면 애플리케이션을 한 번 계측하고 상관관계가 있는 지표 및 추적을 Amazon CloudWatch AWS X-Ray, Amazon OpenSearch Service를 포함한 여러 AWS 모니터링 솔루션으로 전송할 수 있습니다. ADOT와 함께 X-Ray를 사용하려면 두 가지 구성 요소가 필요합니다. X-Ray와 함께 사용할 수 있는 OpenTelemetry SDK와 X-Ray와 함께 사용할 수 있는 OpenTelemetry Collector를 위한AWS 배포판입니다. ADOT Python에는 자동 계측 지원이 포함되어 애플리케이션에서 코드 변경 없이 트레이스를 전송할 수 있습니다.

시작하려면 OpenTelemetry Python용AWS 배포판 설명서를 참조하십시오.

AWS X-Ray 및 기타에서 AWS Distro for OpenTelemetry를 사용하는 방법에 대한 자세한 내용은 AWS Distro for OpenTelemetry 또는 AWS Distro for OpenTelemetry 설명서를 AWS 서비스참조하세요.

언어 지원 및 사용법에 대한 자세한 내용은 Github의AWS 관찰성을 참조하세요.

AWS X-Ray Python용 SDK

Python용 X-Ray SDK는 트레이스 데이터를 생성하고 X-Ray 대몬(daemon)에 보내기 위한 클래스 및 메서드를 제공하는 Python 웹 애플리케이션용 라이브러리 집합입니다. 트레이스 데이터에는 애플리케이션에서 제공하는 수신 HTTP 요청과 애플리케이션이 AWS SDK, HTTP 클라이언트 또는 SQL 데이터베이스 커넥터를 사용하여 다운스트림 서비스에 수행하는 호출에 대한 정보가 포함됩니다. 또한 수동으로 세그먼트를 생성하고 디버그 정보를 주석 및 메타데이터에 추가할 수도 있습니다.

`pip`로 SDK를 다운로드할 수 있습니다.

```
$ pip install aws-xray-sdk
```

> **Note**
>
> Python용 X-Ray SDK는 오픈 소스 프로젝트입니다. 프로젝트를 따르고 GitHub(github.com/aws/aws-xray-sdk-python)에서 문제 및 풀 요청을 제출할 수 있습니다.

Django 또는 Flask를 사용하는 경우 SDK 미들웨어를 애플리케이션에 추가하는 것부터 시작하여 수신 요청을 추적합니다. 미들웨어는 각 트레이스 요청에 대한 세그먼트를 생성하고, 응답이 전송되면 세그먼트를 완료합니다. 세그먼트가 열려 있는 동안에는 SDK 클라이언트의 메서드를 이용해 정보를 세그먼트에 추가하고 하위 세그먼트를 만들어 다운스트림 호출을 트레이스할 수 있습니다. 또한 SDK는 세그먼트가 열려 있는 동안 애플리케이션에서 발생하는 예외를 자동으로 기록합니다. 다른 애플리케이션의 경우 세그먼트를 수동으로 만들 수 있습니다.

계측되는 애플리케이션 또는 서비스에 의해 호출되는 Lambda 함수의 경우, Lambda는 추적 헤더를 읽고 샘플링된 요청을 자동으로 추적합니다. 그 밖의 함수의 경우 수신 요청을 샘플링 및 추적하도록 Lambda를 구성합니다. 어느 경우든, Lambda는 세그먼트를 생성하여 X-Ray SDK에 제공합니다.

> **Note**
>
> Lambda의 X-Ray SDK는 선택 사항입니다. 이를 함수에 사용하지 않는 경우 여전히 서비스 맵에 Lambda 서비스용 노드와 각 Lambda 함수용 노트 하나가 포함됩니다. SDK를 추가하면 함수 코드를 계측하여 Lambda에 의해 기록되는 함수 세그먼트에 하위 세그먼트를 추가할 수 있습니다. 자세한 내용은 AWS Lambda 그리고 AWS X-Ray 섹션을 참조하세요.

Lambda로 계측된 Python 함수의 예는 작업자을 참조하십시오.

다음으로 Python용 X-Ray SDK를 사용하여 애플리케이션의 라이브러리를 패치함으로써 다운스트림 호출을 구성합니다. SDK는 다음 라이브러리를 지원합니다.

지원되는 라이브러리

- `botocore`, `boto3`- AWS SDK for Python (Boto) 클라이언트를 계측합니다.
- `pynamodb` – Amazon DynamoDB 클라이언트의 PynamoDB 버전을 구성합니다.
- `aiobotocore`, `aioboto3` — Python 클라이언트용 비동기 통합 SDK 버전을 구성합니다.
- `requests`, `aiohttp` – 상위 HTTP 클라이언트를 구성합니다.

- `httplib`, `http.client` – 하위 HTTP 클라이언트와 이러한 클라이언트를 사용하는 상위 라이브러리를 구성합니다.
- `sqlite3` – SQLite 클라이언트를 구성합니다.
- `mysql-connector-python` – MySQL 클라이언트를 구성합니다.
- `pg8000` – 순수 Python을 구성합니다.
- `psycopg2` – PostgreSQL 데이터베이스 어댑터를 구성합니다.
- `pymongo` – MongoDB 클라이언트를 구성합니다.
- `pymysql` – MySQL 및 MariaDB 클라이언트에 기반한 PyMySQL을 구성합니다.

애플리케이션이 AWS, SQL 데이터베이스 또는 기타 HTTP 서비스를 호출할 때마다 SDK는 하위 세그먼트에 호출에 대한 정보를 기록합니다. 서비스 내에서 액세스하는 AWS 서비스 리소스는 트레이스 맵에 다운스트림 노드로 표시되므로 개별 연결에서 오류 및 제한 문제를 식별하는 데 도움이 됩니다.

SDK를 사용하기 시작한 후에, 레코더와 미들웨어를 구성하여 SDK 동작을 구성하십시오. 플러그인을 추가해 애플리케이션을 실행하는 컴퓨팅 리소스에 대한 데이터를 기록하고, 샘플링 규칙을 정의해 샘플링 동작을 구성하고, 로그 레벨을 설정해 애플리케이션 로그의 SDK에서 표시되는 정보 수준을 조절할 수 있습니다.

요청에 대한 추가 정보와 애플리케이션이 주석 및 메타데이터에서 하는 작업을 기록합니다. 주석은 필터 표현식과 함께 사용할 수 있도록 인덱싱된 단순한 키 값 쌍이기 때문에, 특정 데이터를 포함한 트레이스를 검색할 수 있습니다. 메타데이터 항목은 제한이 적으며 JSON으로 직렬화할 수 있는 모든 객체와 어레이를 기록할 수 있습니다.

> **주석 및 메타데이터**
>
> 주석 및 메타데이터는 X SDK를 사용하여 세그먼트에 추가하는 임의의 텍스트입니다. 주석은 필터 표현식에서 사용하기 위해 인덱싱됩니다. 메타데이터는 인덱싱되지 않지만 X-Ray 콘솔 또는 API를 사용하여 원시 세그먼트에서 볼 수 있습니다. X-Ray에 대한 읽기 액세스가 부여된 사용자는 누구나 이 데이터를 볼 수 있습니다.

코드에 구성된 클라이언트가 많이 있다면, 구성된 클라이언트로 만든 각 직접 호출의 하위 세그먼트를 대량으로 보관하는 요청 세그먼트 하나를 만들 수 있습니다. 사용자 지정 하위 세그먼트의 클라이언트 호출을 래핑해 하위 세그먼트를 조직하고 그룹화할 수 있습니다. 전체 함수 또는 코드 섹션에 대한 사용자 지정 하위 세그먼트를 생성할 수 있습니다. 그런 다음 상위 세그먼트에 모든 것을 적는 대신 하위 세그먼트에 메타데이터 및 주석을 레코딩할 수 있습니다.

SDK의 클래스 및 메서드에 대한 참조 문서는 AWS X-Ray SDK for Python API Reference를 참조하십시오.

요구 사항

Python용 X-Ray SDK는 다음 언어 및 라이브러리 버전을 지원합니다.

- Python – 2.7, 3.4, 및 최신 버전
- Django – 1.10 이상
- Flask – 0.10 이상
- aiohttp – 2.3.0 이상
- AWS SDK for Python (Boto) – 1.4.0 이상
- botocore – 1.5.0 이상
- enum – 0.4.7 이상, Python 버전 3.4.0 이하인 경우
- jsonpickle – 1.0.0 이상
- setuptools – 40.6.3 이상
- wrapt – 1.11.0 이상

종속성 관리

Python용 X-Ray SDK는 `pip`에서 사용할 수 있습니다.

- 패키지 – `aws-xray-sdk`

SDK를 `requirements.txt` 파일에 종속성으로 추가합니다.

Example requirements.txt

```
aws-xray-sdk==2.4.2
boto3==1.4.4
botocore==1.5.55
Django==1.11.3
```

Elastic Beanstalk를 사용하여 애플리케이션을 배포하는 경우 Elastic Beanstalk은 `requirements.txt`의 모든 패키지를 자동으로 설치합니다.

Python용 X-Ray SDK 구성

Python용 X-Ray SDK에는 전역 레코더를 제공하는 `xray_recorder`라는 클래스가 있습니다. 수신 HTTP 호출에 대해 세그먼트를 생성하는 미들웨어를 사용자 지정하도록 전역 레코더를 구성할 수 있습니다.

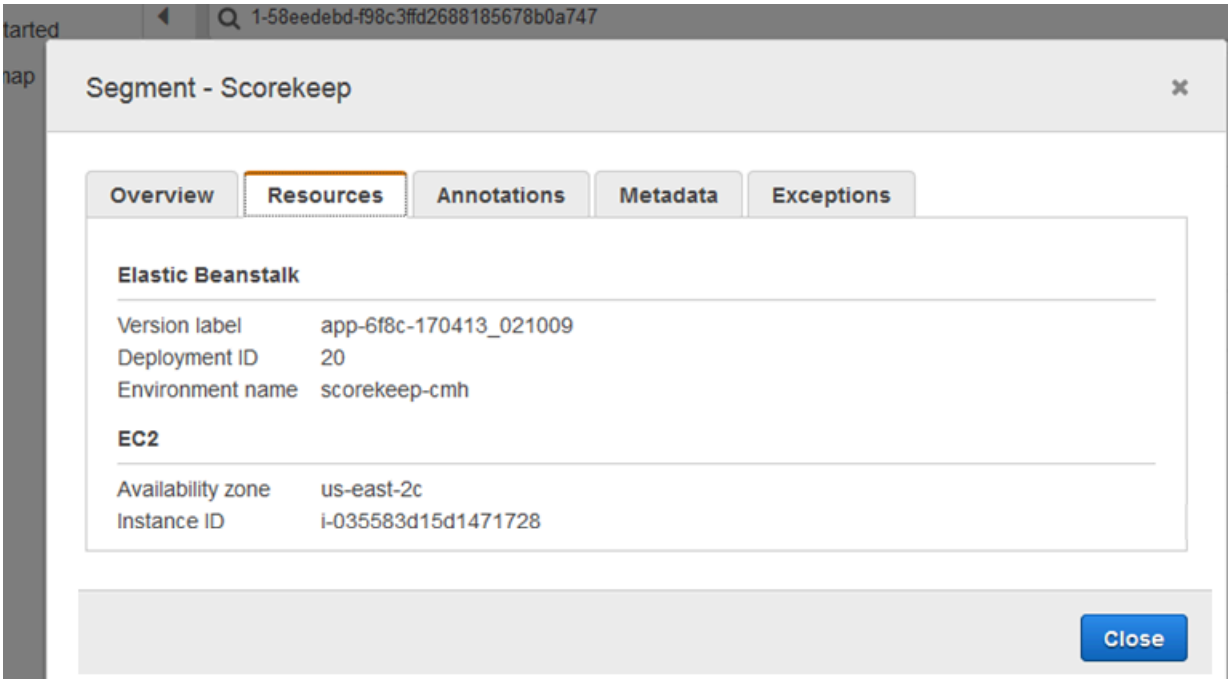
Sections

서비스 플러그인

`plugins`을 사용하여 애플리케이션을 호스팅하는 서비스에 대한 정보를 기록할 수 있습니다.

플러그인

- Amazon EC2 — `EC2Plugin`은 인스턴스 ID, 가용 영역 및 CloudWatch Logs 그룹을 추가합니다.
- Elastic Beanstalk – `ElasticBeanstalkPlugin`이 환경 이름, 버전 레이블 및 배포 ID를 추가합니다.
- Amazon ECS — `ECSPlugin`이 컨테이너 ID를 추가합니다.



플러그인을 사용하려면 `xray_recorder`에서 `configure`를 호출합니다.

```
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**
>
> `plugins`는 튜플로 전달되므로 단일 플러그인을 지정할 때는 반드시 후행 `,`를 포함해야 합니다. 예제: `plugins = ('EC2Plugin',)`

또한 코드에 설정된 값보다 우선하는 환경 변수를 사용하여 레코더를 구성할 수 있습니다.

다운스트림 호출을 레코딩하도록 라이브러리를 패치하기 전에 플러그인을 구성합니다.

SDK는 플러그인 설정을 사용하여 세그먼트에 `origin` 필드를 설정하기도 합니다. 이는 애플리케이션을 실행하는 AWS 리소스 유형을 나타냅니다. 여러 플러그인을 사용하는 경우 SDK는 ElasticBeanstalk> EKS> ECS> EC2 순서로 확인하여 오리진을 결정합니다.

샘플링 규칙

SDK는 X-Ray 콘솔에서 정의하는 샘플링 규칙을 사용하여 기록할 요청을 결정합니다. 기본 규칙은 매초 첫 번째 요청을 추적하고, 모든 서비스에서 추가 요청의 5%를 X-Ray로 추적 전송합니다. X-Ray 콘솔에서 추가 규칙을 생성하여 각 애플리케이션에 대해 기록되는 데이터의 양을 사용자 지정합니다.

SDK는 사용자 지정 규칙을 정의된 순서대로 적용합니다. 요청이 여러 사용자 지정 규칙과 일치하는 경우 SDK는 첫 번째 규칙만 적용합니다.

> **Note**
>
> SDK가 샘플링 규칙을 가져오기 위해 X-Ray에 연결할 수 없는 경우, 매초 첫 번째 요청과 호스트당 추가 요청의 5%에 대한 기본 로컬 규칙으로 되돌아갑니다. 호스트가 샘플링 API를 직접 호출할 수 있는 권한이 없거나, X-Ray 대몬(daemon)에 연결할 수 없을 경우 이러한 상황이 발생할 수 있고 이 대몬(daemon)은 SDK에서 수행한 API 직접 호출에 대한 TCP 프록시 역할을 합니다.

JSON 문서에서 샘플링 규칙을 불러오도록 SDK를 구성할 수도 있습니다. SDK는 X-Ray 샘플링을 사용할 수 없을 경우 로컬 규칙을 백업으로 사용하거나, 로컬 규칙을 전용으로 사용할 수 있습니다.

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
  }
}
```

이 예에서는 하나의 사용자 지정 규칙과 기본 규칙을 정의합니다. 사용자 지정 규칙은 최소 추적 요청 수 없이 5% 샘플링 비율을 `/api/move/` 아래 경로에 적용합니다. 기본 규칙은 매초 최초 요청과 추가 요청의 10%를 추적합니다.

로컬로 규칙의 정의할 때의 단점은 고정 대상이 X-Ray 서비스를 통해 관리되는 대신, 레코더의 각 인스턴스별로 독립적으로 적용된다는 것입니다. 호스트를 많이 배포할수록 고정 속도가 크게 증대하기 때문에 기록되는 데이터의 양을 제어하기가 어려워집니다.

에서는 샘플링 속도를 수정할 수 AWS Lambda없습니다. 구성된 서비스가 함수를 호출하는 경우 해당 서비스에서 샘플링한 요청을 생성한 호출이 Lambda에 의해 기록됩니다. 활성 추적이 활성화되고 트레이스 헤더가 없는 경우 Lambda에서 샘플링 결정을 내립니다.

백업 샘플링 규칙을 구성하려면 다음 예제와 같이 `xray_recorder.configure`를 호출합니다. 이 예제에서 *`rules`*는 규칙의 딕셔너리이거나 샘플링 규칙이 포함된 JSON 파일의 절대 경로입니다.

```
xray_recorder.configure(sampling_rules=rules)
```

로컬 규칙만 사용하려면 `LocalSampler`를 사용하여 레코더를 구성합니다.

```
from aws_xray_sdk.core.sampling.local.sampler import LocalSampler
xray_recorder.configure(sampler=LocalSampler())
```

또한 샘플링을 비활성하하고 모든 수신 요청을 구성하도록 전역 레코더를 구성할 수 있습니다.

Example main.py – 샘플링 비활성화

```
xray_recorder.configure(sampling=False)
```

로깅

SDK는 기본 `WARNING` 로깅 레벨이 있는 Python의 내장 `logging` 모듈을 사용합니다. `aws_xray_sdk` 클래스에 대한 로거 참조를 가져오고 이에 대해 `setLevel`을 호출하여 라이브러리 및 애플리케이션의 나머지에 대해 서로 다른 로그 레벨을 구성합니다.

Example app.py – 로깅

```
logging.basicConfig(level='WARNING')
logging.getLogger('aws_xray_sdk').setLevel(logging.ERROR)
```

디버그 로그를 사용하여 수동으로 하위 세그먼트를 생성할 때 미완료 하위 세그먼트와 같은 문제를 식별할 수 있습니다.

코드의 레코더 구성

`xray_recorder`에 대한 `configure` 메서드에서 추가 설정을 사용할 수 있습니다.

- `context_missing` – 열려 있는 세그먼트가 없는 경우 구성된 코드가 데이터를 기록하려고 할 때 발생하는 예외를 방지하려면 `LOG_ERROR`로 설정합니다.
- `daemon_address` – X-Ray 대몬(daemon) 리스너의 호스트와 포트를 설정합니다.
- `service` – SDK가 세그먼트에 사용하는 서비스 이름을 설정합니다.
- `plugins` - 애플리케이션의 AWS 리소스에 대한 정보를 기록합니다.
- `sampling` – 샘플링을 비활성화하려면 `False`로 설정합니다.
- `sampling_rules` – 샘플링 규칙이 포함된 JSON 파일의 경로를 설정합니다.

Example main.py – 컨텍스트 누락 예외 비활성화

```
from aws_xray_sdk.core import xray_recorder

xray_recorder.configure(context_missing='LOG_ERROR')
```

Django의 레코더 구성

Django 프레임워크를 사용하는 경우 Django `settings.py` 파일을 사용하여 전역 레코더의 옵션을 구성할 수 있습니다.

- `AUTO_INSTRUMENT` (Django 전용) – 기본 제공 데이터베이스 및 템플릿 렌더링 작업을 위한 하위 세그먼트를 기록합니다.
- `AWS_XRAY_CONTEXT_MISSING` – 열려 있는 세그먼트가 없는 경우 구성된 코드가 데이터를 기록하려고 할 때 발생하는 예외를 방지하려면 `LOG_ERROR`로 설정합니다.
- `AWS_XRAY_DAEMON_ADDRESS` – X-Ray 대몬(daemon) 리스너의 호스트와 포트를 설정합니다.
- `AWS_XRAY_TRACING_NAME` – SDK가 세그먼트에 사용하는 서비스 이름을 설정합니다.
- `PLUGINS` - 애플리케이션의 AWS 리소스에 대한 정보를 기록합니다.
- `SAMPLING` – 샘플링을 비활성화하려면 `False`로 설정합니다.
- `SAMPLING_RULES` – 샘플링 규칙이 포함된 JSON 파일의 경로를 설정합니다.

settings.py에서 레코더 구성을 활성화하려면 설치된 앱 목록에 Django 미들웨어를 추가합니다.

Example settings.py – 설치된 앱

```
INSTALLED_APPS = [
    ...
    'django.contrib.sessions',
    'aws_xray_sdk.ext.django',
]
```

이름이 XRAY_RECORDER인 dict에서 사용 가능한 설정을 구성합니다.

Example settings.py – 설치된 앱

```
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,
}
```

환경 변수

환경 변수를 사용하여 Python용 X-Ray SDK를 구성할 수 있습니다. SDK는 다음 변수를 지원합니다:

- AWS_XRAY_TRACING_NAME – SDK가 세그먼트에 사용하는 서비스 이름을 설정합니다. 프로그래밍 방식으로 설정한 서비스 이름을 재정의합니다.
- AWS_XRAY_SDK_ENABLED – false로 설정하면 SDK가 비활성화됩니다. 이 환경 변수가 false로 설정되지 않은 한 SDK는 기본적으로 활성화됩니다.
  - 비활성화될 경우, 글로벌 레코더가 대몬(daemon)으로 전송되지 않는 더미 세그먼트 및 하위 세그먼트를 자동으로 생성하고 자동 패치가 비활성화됩니다. 미들웨어는 글로벌 레코더 위에 래퍼로 기록됩니다. 또한 미들웨어를 통한 세그먼트 및 하위 세그먼트 생성은 모두 더미 세그먼트와 더미 하위 세그먼트가 됩니다.
  - 환경 변수 또는 aws_xray_sdk 라이브러리에서 global_sdk_config 객체와 직접 상호 작용을 통해 AWS_XRAY_SDK_ENABLED 값을 설정합니다. 환경 변수에 대한 설정은 이러한 상호 작용을 재정의합니다.
- AWS_XRAY_DAEMON_ADDRESS – X-Ray 대몬(daemon) 리스너의 호스트와 포트를 설정합니다. 기본적으로 SDK는 추적 데이터(UDP)와 샘플링(TCP) 모두에 127.0.0.1:2000을 사용합니다. 다른 포

트에서 수신 대기하도록 대몬(daemon)을 구성한 경우 또는 다른 호스트에서 실행 중인 경우 이 변수를 사용합니다.

형식

- 동일한 포트 – *address*:*port*
- 다른 포트 – tcp:*address*:*port* udp:*address*:*port*

- AWS_XRAY_CONTEXT_MISSING – 열려 있는 세그먼트가 없는 경우 구성된 코드가 데이터를 기록하려고 할 때 발생하는 예외를 방지하려면 RUNTIME_ERROR로 설정합니다.

  유효한 값

  - RUNTIME_ERROR— 런타임 예외가 발생합니다.
  - LOG_ERROR – 오류를 기록하고 계속합니다 (기본값).
  - IGNORE_ERROR— 오류를 무시하고 계속합니다.

  열린 요청이 없을 때 실행되는 시작 코드 또는 새 스레드를 생성하는 코드에서 계측된 클라이언트를 사용하려고 하는 경우 누락된 세그먼트 또는 하위 세그먼트와 관련된 오류가 발생할 수 있습니다.

환경 변수는 코드에 설정된 값을 재정의합니다.

Python 미들웨어용 X-Ray SDK로 수신 요청 추적하기

미들웨어를 애플리케이션에 추가하고 세그먼트 이름을 구성하면 Python용 X-Ray SDK는 샘플링된 각 요청에 대해 세그먼트를 생성합니다. 이 세그먼트에는 HTTP 요청의 시간, 메서드 및 배치가 포함됩니다. 추가로 구성하면 이 세그먼트의 하위 세그먼트가 생성됩니다.

Python용 X-Ray SDK는 수신 HTTP 요청을 구성하기 위한 다음 미들웨어를 지원합니다.

- Django
- Flask
- Bottle

> **Note**
>
> AWS Lambda 함수의 경우 Lambda는 샘플링된 각 요청에 대해 세그먼트를 생성합니다. 자세한 내용은 AWS Lambda 그리고 AWS X-Ray 섹션을 참조하세요.

Lambda로 계측된 Python 함수의 예는 작업자을 참조하십시오.

다른 프레임워크의 스크립트 또는 Python 애플리케이션의 경우, 세그먼트를 수동으로 만들 수 있습니다.

각 세그먼트에는 서비스 맵 안에서 애플리케이션을 식별하는 이름이 있습니다. 이 세그먼트의 이름이 정적으로 지정되도록 하거나, 수신 요청의 호스트 헤더를 기반으로 SDK가 동적으로 이름을 지정하도록 구성할 수 있습니다. 동적 이름 지정을 사용하면 요청의 도메인 이름에 따라 그룹을 추적하고 이름이 예상 패턴과 일치하지 않을 경우(예: 호스트 헤더가 위조된 경우) 기본 이름을 적용할 수 있습니다.

> **전달된 요청**
>
> 로드 밸런서 또는 기타 중개자가 애플리케이션으로 요청을 전달하는 경우, X-Ray는 IP 패킷 내 소스 IP가 아니라 요청의 `X-Forwarded-For` 헤더로부터 클라이언트 IP를 가져옵니다. 전달된 요청에 대해 기록된 클라이언트 IP는 위조될 수 있으므로 신뢰하면 안 됩니다.

요청이 전달되면 SDK는 세그먼트에 추가 필드를 설정하여 이를 나타냅니다. 세그먼트에 `x_forwarded_for`로 설정된 `true` 필드가 포함된 경우 클라이언트 IP는 HTTP 요청의 `X-Forwarded-For` 헤더로부터 가져옵니다.

미들웨어는 다음 정보를 포함하는 `http` 블록으로 각 수신 요청에 대한 세그먼트를 생성합니다.

- HTTP 메서드 – GET, POST, PUT, DELETE 등.
- 클라이언트 주소 – 요청을 전송한 클라이언트의 IP 주소.
- 응답 코드 – 완료된 요청의 HTTP 응답 코드.
- 시간 – 시작 시간(요청 수신) 및 종료 시간(응답 전송).
- 유저 에이전트 — 요청에서 가져온 `user-agent`입니다.
- 콘텐츠 길이 — 응답의 `content-length`입니다.

Sections

- 애플리케이션(Django)에 미들웨어 추가
- 애플리케이션(Flask)에 미들웨어 추가
- 애플리케이션(Bottle)에 미들웨어 추가
- 수동으로 Python 코드 구성
- 세그먼트 이름 지정 전략 구성

애플리케이션(Django)에 미들웨어 추가

`MIDDLEWARE` 파일에서 `settings.py` 목록에 미들웨어를 추가합니다. 기타 미들웨어에서 실패한 요청이 레코딩되도록 X-Ray 미들웨어가 `settings.py` 파일에서 첫 번째 줄에 있어야 합니다.

Example settings.py - Python 미들웨어용 X-Ray SDK

```
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'
]
```

`settings.py` 파일의 `INSTALLED_APPS` 목록에 X-Ray SDK Django 앱을 추가하세요. 이렇게 하면 앱을 시작하는 동안 X-Ray 레코더를 구성할 수 있습니다.

Example settings.py - Python Django 앱용 X-Ray SDK

```
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',
]
```

`settings.py 파일`에서 세그먼트 이름을 구성합니다.

Example settings.py – 세그먼트 이름

```
XRAY_RECORDER = {
    'AWS_XRAY_TRACING_NAME': 'My application',
    'PLUGINS': ('EC2Plugin',),
}
```

이는 X-Ray 레코더가 기본 샘플링 속도로 Django 애플리케이션에서 제공하는 요청을 추적하도록 지시합니다. 사용자 지정 샘플링 규칙을 적용하거나 기타 설정을 변경하도록 Django 설정 파일의 레코더를 구성할 수 있습니다.

> **Note**
>
> `plugins`는 튜플로 전달되므로 단일 플러그인을 지정할 때는 반드시 후행 `,`를 포함해야 합니다. 예제: `plugins = ('EC2Plugin',)`

애플리케이션(Flask)에 미들웨어 추가

Flask 애플리케이션을 구성하려면 먼저 `xray_recorder`에서 세그먼트 이름을 구성해야 합니다. 그런 다음 `XRayMiddleware` 함수를 사용하여 Flask 애플리케이션을 코드로 패치합니다.

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)
```

이는 X-Ray 레코더가 기본 샘플링 속도로 Flask 애플리케이션에서 제공하는 요청을 추적하도록 지시합니다. 사용자 지정 샘플링 규칙을 적용하거나 기타 설정을 변경하도록 레코더를 코드로 구성할 수 있습니다.

애플리케이션(Bottle)에 미들웨어 추가

Bottle 애플리케이션을 구성하려면 먼저 `xray_recorder`에서 세그먼트 이름을 구성해야 합니다. 그런 다음 `XRayMiddleware` 함수를 사용하여 Bottle 애플리케이션을 코드로 패치합니다.

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))
```

이는 X-Ray 레코더가 기본 샘플링 속도로 Bottle 애플리케이션에서 제공하는 요청을 추적하도록 지시합니다. 사용자 지정 샘플링 규칙을 적용하거나 기타 설정을 변경하도록 레코더를 코드로 구성할 수 있습니다.

수동으로 Python 코드 구성

Django 또는 Flask를 사용하지 않는 경우 세그먼트를 수동으로 만들 수 있습니다. 수신되는 각 요청에 대해 세그먼트를 생성하거나 패치된 HTTP 또는 AWS SDK 클라이언트 주위에 세그먼트를 생성하여 레코더가 하위 세그먼트를 추가할 컨텍스트를 제공할 수 있습니다.

Example main.py – 수동 구성

```
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()
```

세그먼트 이름 지정 전략 구성

AWS X-Ray 는 서비스 이름을 사용하여 애플리케이션을 식별하고 애플리케이션이 사용하는 다른 애플리케이션, 데이터베이스, 외부 APIs 및 AWS 리소스와 구분합니다. X-Ray SDK는 수신 요청에 대한 세그먼트를 생성할 때 해당 세그먼트의 이름 필드에 애플리케이션의 서비스 이름을 기록합니다.

X-Ray SDK는 HTTP 요청 헤더의 호스트 이름 뒤에 세그먼트를 지정할 수 있습니다. 그러나 이 헤더가 위조되면 서비스 맵에 예기치 않은 노드가 발생할 수 있습니다. 위조된 호스트 헤더가 포함된 요청으로 인해 SDK가 잘못된 세그먼트 이름을 지정하는 현상을 방지하려면 들어오는 요청의 기본 이름을 지정해야 합니다.

애플리케이션이 여러 도메인의 요청을 처리하는 경우, 동적 이름 지정 전략을 사용하여 이를 세그먼트 이름에 반영하도록 SDK를 구성할 수 있습니다. 동적 이름 지정 전략을 사용하면 SDK가 예상 패턴과 일치하는 요청에 호스트 이름을 사용하고, 그렇지 않은 요청에 기본 이름을 적용할 수 있습니다.

예를 들어, 하나의 애플리케이션이 세 개의 하위 도메인 (`www.example.com`, `api.example.com`, `static.example.com`) 에 요청을 전송할 수 있습니다. `*.example.com` 패턴으로 동적 이름 지정 전략을 사용하여 각 하위도메인의 세그먼트를 서로 다른 이름으로 표시하면 서비스 맵에 서비스 노드가 세 개 생깁니다. 이 패턴에 맞지 않는 호스트 이름의 요청이 애플리케이션에 수신되면, 사용자가 지정한 대체 이름의 네 번째 노드가 서비스 맵에 표시됩니다.

모든 요청 세그먼트에 같은 이름을 사용하려면 이전 섹션에 나온 것처럼 레코더를 구성할 때 애플리케이션 이름을 지정합니다.

동적 이름 지정 전략은 호스트 이름이 일치해야 하는 패턴 및 HTTP 요청의 호스트 이름이 패턴과 일치하지 않는 경우 사용할 기본 이름을 정의합니다. Django에서 동적으로 세그먼트에 이름을 지정하려면 `DYNAMIC_NAMING` 설정을 settings.py 파일에 추가합니다.

Example settings.py – 동적 이름 지정

```
XRAY_RECORDER = {
    'AUTO_INSTRUMENT': True,
    'AWS_XRAY_TRACING_NAME': 'My application',
    'DYNAMIC_NAMING': '*.example.com',
    'PLUGINS': ('ElasticBeanstalkPlugin', 'EC2Plugin')
}
```

패턴에서 '*'를 사용하여 문자열을 일치시키거나 '?'를 사용하여 단일 문자를 일치시킬 수 있습니다. Flask의 경우 레코더를 코드로 구성합니다.

Example main.py – 세그먼트 이름

```
from aws_xray_sdk.core import xray_recorder
xray_recorder.configure(service='My application')
xray_recorder.configure(dynamic_naming='*.example.com')
```

> **Note**
>
> 코드에 정의한 기본 서비스 이름을 `AWS_XRAY_TRACING_NAME` 환경 변수를 사용하여 재정의할 수 있습니다.

다운스트림 호출 구성을 위해 라이브러리 패치

다운스트림 호출을 구성하려면 Python용 X-Ray SDK를 사용하여 애플리케이션이 사용하는 라이브러리를 패치합니다. Python용 X-Ray SDK는 다음 라이브러리를 패치할 수 있습니다.

지원되는 라이브러리

- `botocore`, `boto3`- AWS SDK for Python (Boto) 클라이언트를 계측합니다.
- `pynamodb` – Amazon DynamoDB 클라이언트의 PynamoDB 버전을 구성합니다.
- `aiobotocore`, `aioboto3` — Python 클라이언트용 비동기 통합 SDK 버전을 구성합니다.
- `requests`, `aiohttp` – 상위 HTTP 클라이언트를 구성합니다.
- `httplib`, `http.client` – 하위 HTTP 클라이언트와 이러한 클라이언트를 사용하는 상위 라이브러리를 구성합니다.
- `sqlite3` – SQLite 클라이언트를 구성합니다.
- `mysql-connector-python` – MySQL 클라이언트를 구성합니다.
- `pg8000` – 순수 Python을 구성합니다.
- `psycopg2` – PostgreSQL 데이터베이스 어댑터를 구성합니다.
- `pymongo` – MongoDB 클라이언트를 구성합니다.
- `pymysql` – MySQL 및 MariaDB 클라이언트에 기반한 PyMySQL을 구성합니다.

패치된 라이브러리를 사용하면 Python용 X-Ray SDK는 직접 호출에 대한 하위 세그먼트를 생성하고 요청 및 응답의 정보를 기록합니다. SDK가 SDK 미들웨어 또는 AWS Lambda에서 하위 세그먼트를 생성하려면 세그먼트를 사용할 수 있어야 합니다.

> **Note**
>
> SQLAlchemy ORM을 사용하는 경우, SQLAlchemy 세션의 SDK 버전과 쿼리 클래스를 가져와서 SQL 쿼리를 구성할 수 있습니다. 지침은 SQLAlchemy ORM 사용을 참조하십시오.

사용 가능한 모든 라이브러리를 패치하려면 `aws_xray_sdk.core`에서 `patch_all` 함수를 사용합니다. `httplib` 및 `urllib`와 같은 일부 라이브러리는 `patch_all(double_patch=True)`을 호출하여 이중 패치 적용을 활성화해야 할 수 있습니다 .

Example main.py – 지원되는 모든 라이브러리 패치

```
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()
```

단일 라이브러리에 패치를 적용하려면 라이브러리 이름의 튜플을 사용하여 patch를 호출합니다. 이를 위해서는 단일 요소 목록을 제공해야 합니다.

Example main.py – 특정 라이브러리 패치

```
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**
>
> 라이브러리를 패치하는 데 사용한 키가 라이브러리 이름과 맞지 않는 경우도 있습니다. 일부 키는 하나 이상의 라이브러리에 대해 별칭 역할을 합니다.
>
> 라이브러리 별칭
>
> - `httplib` – `httplib` 및 `http.client`
> - `mysql` – `mysql-connector-python`

비동기 작업에 대한 컨텍스트 추적

asyncio 통합 라이브러리의 경우 또는 비동기 함수에 대한 하위 세그먼트를 생성하려면 비동기 컨텍스트를 사용하여 Python용 X-Ray SDK도 구성해야 합니다. AsyncContext 클래스를 가져와서 이 클래스의 인스턴스를 X-Ray 레코더에 전달합니다.

> **Note**
>
> AIOHTTP와 같은 웹 프레임워크 지원 라이브러리는 aws_xray_sdk.core.patcher 모듈을 통해 처리되지 않습니다. 이들은 지원되는 라이브러리의 patcher 카탈로그에 나타나지 않습니다.

Example main.py – aioboto3 패치

```
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)
```

Python용 X-Ray AWS SDK를 사용하여 SDK 호출 추적

애플리케이션이를 호출 AWS 서비스 하여 데이터를 저장하거나, 대기열에 쓰거나, 알림을 보내면 Python용 X-Ray SDK는 하위 세그먼트에서 다운스트림으로 호출을 추적합니다. 해당 서비스(예: Amazon S3 버킷 또는 Amazon SQS 대기열) 내에서 액세스하는 추적 AWS 서비스 및 리소스는 X-Ray 콘솔의 추적 맵에 다운스트림 노드로 표시됩니다.

Python용 X-Ray SDK는 라이브러리를 패치할 때 모든 AWS SDK 클라이언트를 자동으로 계측합니다. botocore 개별 클라이언트를 구성할 수는 없습니다.

모든 서비스의 경우, X-Ray 콘솔에서 호출된 API의 이름을 볼 수 있습니다. 서비스 하위 집합에 대해서는 X-Ray SDK가 세그먼트에 정보를 추가하여 서비스 맵에서 추가 세분화를 제공합니다.

예를 들어 계측된 DynamoDB 클라이언트에서 직접 호출을 생성하는 경우 SDK가 특정 테이블을 대상으로 한 직접 호출에 대해 테이블 이름을 세그먼트에 추가합니다. 콘솔에서, 각 테이블이 개별 노드로 서비스 맵에 표시되고, 특정 테이블을 대상으로 하지 않은 직접 호출에 대해 일반 DynamoDB 노드가 표시됩니다.

Example 항목을 저장하기 위한 DynamoDB 직접 호출에 대한 하위 세그먼트

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

명명된 리소스에 액세스할 때 다음 서비스를 호출할 경우 서비스 맵에 추가 노드가 생성됩니다. 특정 리소스를 대상으로 하지 않는 경우 서비스를 직접 호출하면 해당 서비스에 대한 일반 노드가 생성됩니다.

- Amazon DynamoDB – 테이블 이름
- Amazon Simple Storage Service –버킷 및 키 이름
- Amazon Simple Queue Service – 대기열 이름

Python용 X-Ray SDK를 사용하여 다운스트림 HTTP 웹 서비스에 대한 호출 추적하기

애플리케이션이 마이크로서비스 또는 퍼블릭 HTTP API를 호출하면 Python용 X-Ray SDK를 사용하여 해당 호출을 구성하고 API를 서비스 그래프에 다운스트림 서비스로 추가할 수 있습니다.

HTTP 클라이언트를 구성하려면 발신 호출을 생성할 때 사용하는 라이브러리를 패치합니다. `requests` 또는 Python의 내장 HTTP 클라이언트를 사용하는 경우 더 이상 수행할 작업이 없습니다. `aiohttp`의 경우 비동기 컨텍스트를 사용하여 레코더를 구성합니다.

`aiohttp` 3의 클라이언트 API를 사용하면, SDK에서 제공하는 추적 구성의 인스턴스로 `ClientSession`의 클라이언트 API도 함께 구성해야 합니다.

Example **`aiohttp`** 3 클라이언트 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()
```

다운스트림 웹 API에 대한 직접 호출을 계측할 때 Python용 X-Ray SDK가 HTTP 요청 및 응답에 대한 정보가 포함된 하위 세그먼트를 기록합니다. X-Ray는 하위 세그먼트를 사용하여 원격 API에 대해 추정된 세그먼트를 생성합니다.

Example 다운스트림 HTTP 호출에 대한 하위 세그먼트

```
{
  "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 다운스트림 HTTP 호출에 대한 추정된 세그먼트

```
{
  "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
}
```

Python용 X-Ray SDK를 사용하여 사용자 지정 하위 세그먼트 생성하기

하위 세그먼트는 추적의 세그먼트를 확장하여 요청을 처리하기 위해 완료된 작업에 대한 세부 정보를 표시합니다. 계측되는 클라이언트에서 직접 호출할 때마다, X-Ray SDK는 하위 세그먼트 안에 생성된 정보를 기록합니다. 추가 하위 세그먼트를 생성하여 다른 하위 세그먼트를 그룹화하거나, 코드 섹션의 성능을 평가하거나, 주석 및 메타데이터를 기록할 수 있습니다.

하위 세그먼트를 관리하려면 `begin_subsegment` 및 `end_subsegment` 메서드를 사용합니다.

Example main.py – 사용자 지정 하위 세그먼트

```
from aws_xray_sdk.core import xray_recorder

subsegment = xray_recorder.begin_subsegment('annotations')
subsegment.put_annotation('id', 12345)
xray_recorder.end_subsegment()
```

동기식 함수에 대한 하위 세그먼트를 생성하려면 `@xray_recorder.capture` 데코레이터를 사용합니다. 하위 세그먼트에 대한 이름을 캡처 함수에 전달하거나 그대로 두고 함수 이름을 사용할 수 있습니다.

Example main.py – 함수 하위 세그먼트

```
from aws_xray_sdk.core import xray_recorder

@xray_recorder.capture('## create_user')
def create_user():
...
```

비동기 함수의 경우 `@xray_recorder.capture_async` 데코레이터를 사용하며, 비동기 컨텍스트를 레코더에 전달합니다.

Example main.py – 비동기 함수 하위 세그먼트

```
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()
```

하위 세그먼트를 세그먼트 또는 다른 하위 세그먼트 내에서 생성하면 Python용 X-Ray SDK가 해당 하위 세그먼트에 대해 ID를 생성하고 시작 시간 및 종료 시간을 기록합니다.

Example 메타데이터가 포함된 하위 세그먼트

```
"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"
    }
  },
```

Python용 X-Ray SDK로 세그먼트에 주석 및 메타데이터 추가하기

주석 및 메타데이터와 함께 요청, 환경 또는 애플리케이션에 대한 추가 정보를 기록할 수 있습니다. X-Ray SDK에서 생성하는 세그먼트 또는 사용자가 생성하는 사용자 지정 하위 세그먼트에 주석 및 메타데이터를 추가할 수 있습니다.

주석은 문자열, 숫자 또는 부울 값과 결합한 키-값 페어입니다. 주석은 필터 표현식에서 사용하기 위해 인덱싱됩니다. 주석은 콘솔의 트레이스를 그룹화할 때 사용할 데이터를 기록하거나 `GetTraceSummaries` API를 직접 호출할 때 사용하세요.

메타데이터는 객체 및 목록을 포함한 모든 유형의 값을 가질 수 있는 키-값 페어지만, 필터 표현식에 사용할 수 있도록 인덱싱되지는 않습니다. 트레이스에 저장하고 싶지만 검색에는 사용하지 않을 추가 데이터는 메타데이터를 사용하여 기록하십시오.

세그먼트에는 주석과 메타데이터 외에 사용자 ID 문자열도 기록할 수 있습니다. 사용자 ID는 세그먼트의 별도 필드에 기록되면 검색용으로 인덱스되지 않습니다.

Sections

Python용 X-Ray SDK로 주석 기록하기

주석을 사용하여 검색용으로 인덱싱할 정보를 세그먼트나 하위 세그먼트에 기록하십시오.

주석 요구 사항

- 키 - X-Ray 주석의 키는 최대 500자의 영숫자를 포함할 수 있습니다. 점이나 마침표(.) 이외의 공백이나 기호를 사용할 수 없습니다.
- 값 - X-Ray 주석의 값은 최대 1,000자의 유니코드 문자를 포함할 수 있습니다.
- 주석 수 - 트레이스당 최대 50개의 주석을 사용할 수 있습니다.

주석 기록 방법

1. `xray_recorder`에서 현재 세그먼트나 하위 세그먼트의 참조를 가져오십시오.

```
from aws_xray_sdk.core import xray_recorder
```

```
...
document = xray_recorder.current_segment()
```

or

```
from aws_xray_sdk.core import xray_recorder
...
document = xray_recorder.current_subsegment()
```

2. 문자열 키, 부울, 숫자 또는 문자열 값으로 put_annotation을 직접 호출합니다.

```
document.put_annotation("mykey", "my value");
```

다음 예에서는 점이 포함된 문자열 키와 부울, 숫자 또는 문자열 값을 사용하여 putAnnotation을 직접 호출하는 방법을 보여줍니다.

```
document.putAnnotation("testkey.test", "my value");
```

또는 put_annotation에서 xray_recorder 방법을 사용할 수 있습니다. 이 방법은 현재의 하위 세그먼트나 열려있는 하위 세그먼트가 없을 경우 해당 세그먼트에서 레코드 주석을 기록합니다.

```
xray_recorder.put_annotation("mykey", "my value");
```

SDK는 세그먼트 문서의 annotations 객체에 주석을 키-값 페어로 기록합니다. 같은 키로 put_annotation을 두 번 직접 호출하면 같은 세그먼트나 하위 세그먼트에 기록했던 값을 덮어씁니다.

특정 값을 포함한 주석이 있는 트레이스를 찾으려면 annotation[*key*] 키워드를 필터 표현식에 사용하십시오.

Python용 X-Ray SDK로 메타데이터 기록하기

> **Warning**
>
> 순환 참조가 있는 객체를 Python용 X-Ray SDK의 메타데이터 값으로 추가하지 마세요. 이러한 객체는 JSON으로 직렬화할 수 없으며 SDK에서 무한 루프를 생성할 수 있습니다. 또한 성능 문제를 방지하기 위해 크고 복잡한 객체를 메타데이터로 추가하지 마세요.

메타데이터를 이용해 검색용으로 인덱싱하지 않아도 되는 정보를 세그먼트나 하위 세그먼트에 기록하십시오. 메타데이터 값은 문자열, 숫자, 부울 또는 JSON 객체나 어레이에 직렬화할 수 있는 모든 객체가 될 수 있습니다.

메타데이터 기록 방법

1. xray_recorder에서 현재 세그먼트나 하위 세그먼트의 참조를 가져오십시오.

```
from aws_xray_sdk.core import xray_recorder
...
document = xray_recorder.current_segment()
```

or

```
from aws_xray_sdk.core import xray_recorder
...
document = xray_recorder.current_subsegment()
```

2. 문자열 키, 부울, 숫자, 문자열 또는 객체 값 및 문자열 네임스페이스와 함께 put_metadata를 직접 호출합니다.

```
document.put_metadata("my key", "my value", "my namespace");
```

or

키와 값만 이용해 put_metadata를 직접 호출합니다.

```
document.put_metadata("my key", "my value");
```

또는 put_metadata에서 xray_recorder 방법을 사용할 수 있습니다. 이 방법은 현재의 하위 세그먼트나 열려있는 하위 세그먼트가 없을 경우 해당 세그먼트에서 메타데이터를 기록합니다.

```
xray_recorder.put_metadata("my key", "my value");
```

네임스페이스를 지정하지 않으면, SDK는 default를 사용합니다. 같은 키로 put_metadata을 두 번 직접 호출하면 같은 세그먼트나 하위 세그먼트에 기록했던 값을 덮어씁니다.

Python용 X-Ray SDK로 사용자 ID 기록하기

사용자 ID를 요청 세그먼트에 기록하여 요청을 보낸 사용자를 식별합니다.

사용자 ID 기록 방법

1. `xray_recorder`에서 현재 세그먼트에 대한 참조를 가져옵니다.

```
from aws_xray_sdk.core import xray_recorder
...
document = xray_recorder.current_segment()
```

2. 요청을 보낸 사용자의 문자열 ID로 `setUser`를 직접 호출합니다.

```
document.set_user("U12345");
```

컨트롤러에서 `set_user`를 직접 호출하면 애플리케이션이 요청을 처리하는 순간부터 사용자 ID를 기록할 수 있습니다.

사용자 ID의 트레이스를 찾으려면, `user` 키워드를 필터 표현식에 적용하십시오.

서버리스 환경에 배포된 웹 프레임워크 구성

Python용 AWS X-Ray SDK는 서버리스 애플리케이션에 배포된 웹 프레임워크 계측을 지원합니다. 서버리스는 운영상의 책임을 AWS로 전환하여 민첩성과 혁신을 높일 수 있도록 하는 클라우드의 네이티브 아키텍처입니다.

서버리스 아키텍처는 서버를 고려하지 않고 애플리케이션 및 서비스를 구축하고 실행할 수 있게 해주는 소프트웨어 애플리케이션입니다. 서버 또는 클러스터 프로비저닝, 패치 적용, 운영 체제 유지 관리 및 용량 프로비저닝과 같은 인프라 관리 작업을 덜어냅니다. 거의 모든 유형의 애플리케이션 또는 백엔드 서비스를 서버리스 솔루션으로 구축할 수 있으며, 애플리케이션을 고가용성으로 실행하고 확장하는 데 필요한 모든 사항이 자동으로 처리됩니다.

이 자습서에서는 서버리스 환경에 배포된 Flask 또는 Django와 같은 웹 프레임워크 AWS X-Ray 에서를 자동으로 계측하는 방법을 보여줍니다. 애플리케이션의 X-Ray 계측을 사용하면 Amazon API Gateway에서 AWS Lambda 함수를 통해 수행된 모든 다운스트림 호출과 애플리케이션이 수행한 발신 호출을 볼 수 있습니다.

Python용 X-Ray SDK는 다음 Python 애플리케이션 프레임워크를 지원합니다.

- Flask 버전 0.8 이상
- Django 버전 1.0 이상

이 튜토리얼에서는 Lambda에 배포되고 API Gateway에서 호출되는 서버리스 애플리케이션 예시를 개발합니다. 이 튜토리얼에서는 Zappa를 사용하여 애플리케이션을 Lambda에 자동으로 배포하고 API 게이트웨이 엔드포인트를 구성합니다.

사전 조건

- Zappa
- Python – 버전 2.7 또는 3.6.
- AWS CLI - AWS CLI 가 계정으로 구성되어 있고 애플리케이션을 배포할 AWS 리전 것인지 확인합니다.
- Pip
- Virtualenv

단계 1: 환경 조성

이 단계에서는 `virtualenv`를 사용하여 애플리케이션을 호스팅할 가상 환경을 만듭니다.

1. 를 사용하여 애플리케이션의 디렉터리를 AWS CLI생성합니다. 그런 다음 새 디렉터리로 변경합니다.

   ```
   mkdir serverless_application
   cd serverless_application
   ```

2. 다음에는 새 디렉터리에서 가상 환경을 생성합니다. 다음 명령을 사용하여 활성화합니다.

   ```
   # Create our virtual environment
   virtualenv serverless_env

   # Activate it
   source serverless_env/bin/activate
   ```

3. 환경에 X-Ray, Flask, Zappa 및 Requests 라이브러리를 설치합니다.

   ```
   # Install X-Ray, Flask, Zappa, and Requests into your environment
   pip install aws-xray-sdk flask zappa requests
   ```

4. `serverless_application` 디렉터리에 애플리케이션 코드를 추가합니다. 이 예제에서는 Flasks의 Hello World 예제를 빌드합니다.

   `serverless_application` 디렉터리에서 `my_app.py`라는 파일 만듭니다. 그런 다음 텍스트 편집기를 사용하여 다음 명령을 추가합니다. 이 애플리케이션은 Requests 라이브러리를 계측하고, Flask 애플리케이션의 미들웨어를 패치하고, 엔드포인트 `'/'`를 엽니다.

```
# 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
```

2단계: Zappa 환경 생성 및 배포

이 단계에서는 Zappa를 사용하여 자동으로 API 게이트웨이 엔드포인트를 구성한 후 Lambda에 배포합니다.

1. `serverless_application` 디렉터리에서 Zappa를 실행합니다. 이 예제에서는 기본 설정을 사용했지만, 기본 설정을 사용자 지정할 경우 Zappa가 구성 지침을 표시합니다.

```
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. X-Ray 활성화하기 `zappa_settings.json` 파일을 열고 예제와 비슷한지 확인합니다.

```
{
    "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. 구성 파일의 한 항목으로 `"xray_tracing": true`를 추가합니다.

```
{
    "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. 애플리케이션을 배포합니다. 그러면 자동으로 API 게이트웨이 엔드포인트가 구성되고 코드가 Lambda로 업로드됩니다.

```
zappa deploy
```

```
...
Deploying API Gateway..
Deployment complete!: https://**********.execute-api.us-west-2.amazonaws.com/dev
```

3단계: API Gateway에 X-Ray 추적 활성화하기

이 단계에서는 API 게이트웨이 콘솔과 상호 작용을 통해 X-Ray 추적을 활성화합니다.

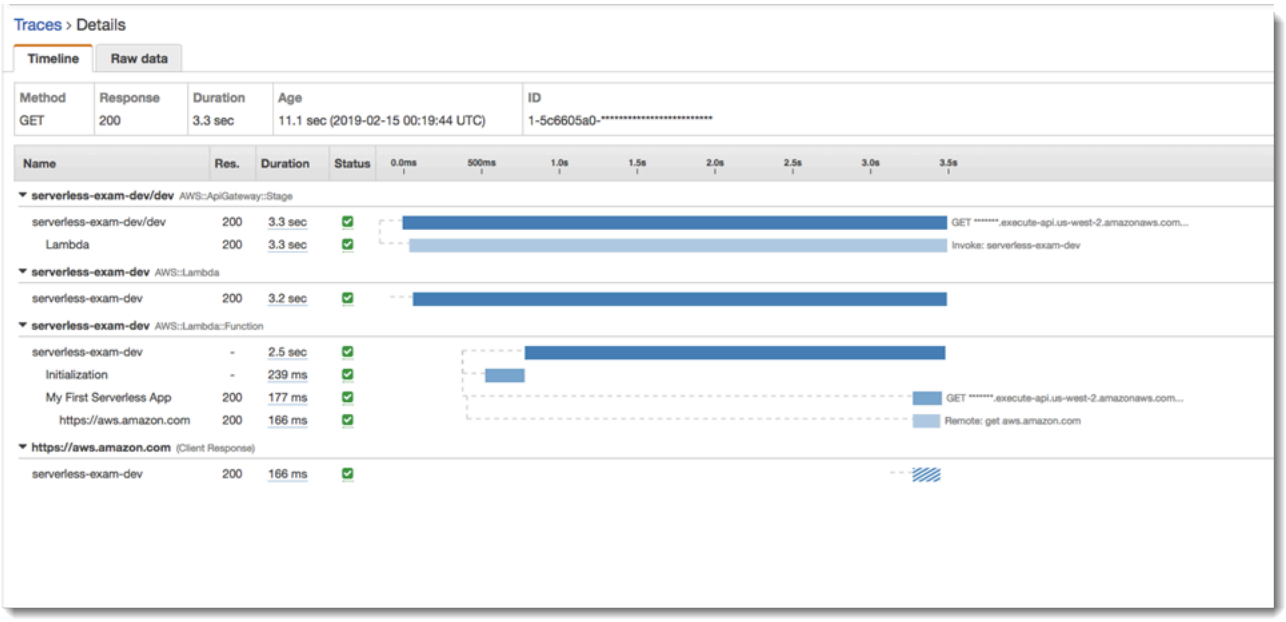
1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/apigateway/:// https://https://https://://https://://https://://https://://https://:/://https://://https://:/:/:/://
2. 새로 생성한 API를 찾습니다. 이 URL은 `serverless-exam-dev`과 유사해야 합니다.
3. 단계를 선택합니다.
4. 배포 단계의 이름을 선택합니다. 기본값은 `dev`입니다.
5. Logs/Tracing(로그/추적) 탭에서 Enable X-Ray Tracing(X-Ray 추적 활성화) 확인란을 선택합니다.
6. 변경 사항 저장을 선택합니다.
7. 브라우저에서 엔드포인트에 액세스합니다. 예제 `Hello World` 애플리케이션을 사용한 경우 다음과 같이 표시되어야 합니다.

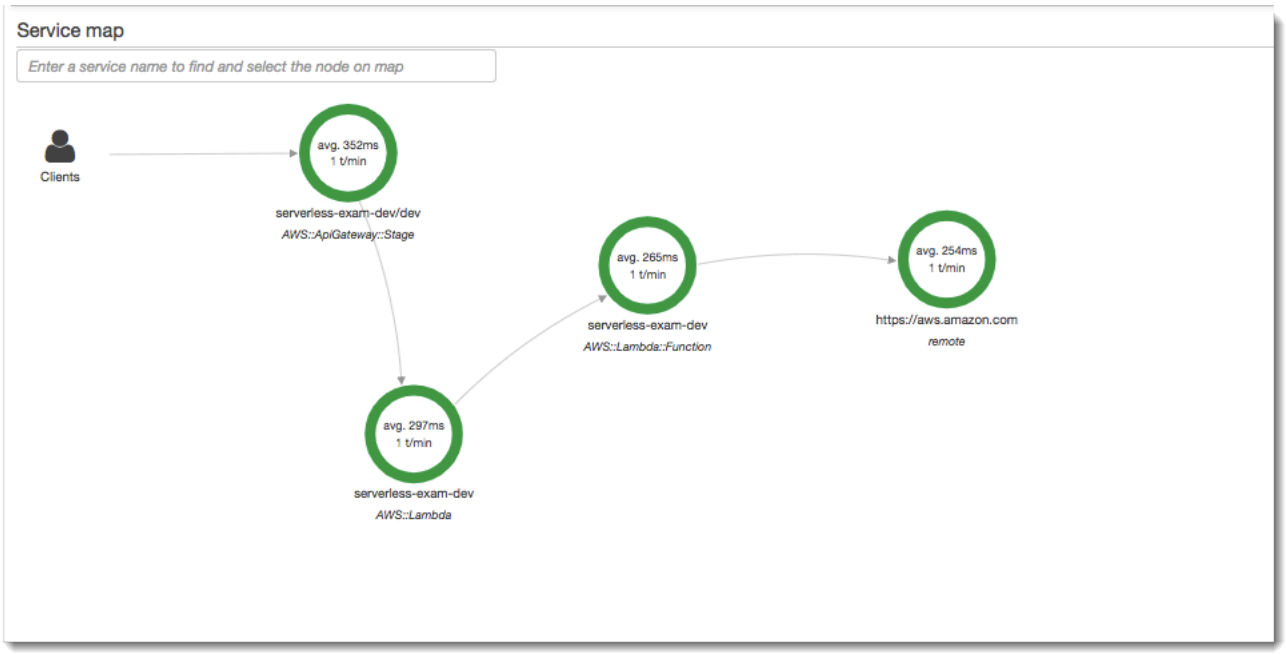
```
"Hello, World: https://aws.amazon.com/"
```

4단계: 생성된 추적 보기

이 단계에서는 X-Ray 콘솔과 상호 작용을 통해 예제 애플리케이션이 생성하는 추적을 확인합니다. 추적 분석에 대한 자세한 내용은 서비스 맵 보기를 참조하십시오.

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/xray/home://https:// https://https://https://://https://://https://:/://https://:/:/:/://www.
2. API 게이트웨이, Lambda 함수 및 Lambda 컨테이너에 의해 생성된 세그먼트를 봅니다.
3. Lambda 함수 세그먼트 아래에서 하위 세그먼트 `My First Serverless App`을 봅니다. 그 다음에 두 번째 하위 세그먼트 `https://aws.amazon.com`이 나옵니다.
4. 초기화 시, Lambda가 세 번째 하위 세그먼트 `initialization`도 생성할 수 있습니다.





5단계: 정리

원치 않는 비용이 누적되지 않도록 항상 더 이상 사용하지 않는 리소스는 종료하십시오. 이 자습서에서 알 수 있듯이 Zappa와 같은 도구는 서버리스 배포를 간소화합니다.

Lambda, API 게이트웨이 및 Amazon S3에서 애플리케이션을 제거하려면 AWS CLI를 사용하여 프로젝트 디렉터리에서 다음 명령을 실행합니다.

```
zappa undeploy dev
```

다음 단계

AWS 클라이언트를 추가하고 X-Ray로 계측하여 애플리케이션에 더 많은 기능을 추가합니다. 서버리스 온 AWS에서 서버리스 컴퓨팅 옵션에 대해 자세히 알아보십시오.

.NET을 사용한 작업

.NET 애플리케이션을 계측하여 트레이스를 X-Ray로 보내는 방법에는 두 가지가 있습니다:

- AWS Distro for OpenTelemetry .NET - AWS Distro for OpenTelemetry Collector를 통해 상관관계가 있는 지표 및 트레이스를 Amazon CloudWatch AWS X-Ray및 Amazon OpenSearch Service를 포함한 여러 AWS 모니터링 솔루션으로 전송하기 위한 오픈 소스 라이브러리 세트를 제공하는 AWS 배포입니다.
- AWS X-Ray .NET용 SDK - X-Ray 데몬을 통해 트레이스를 생성하고 X-Ray로 전송하기 위한 라이브러리 세트입니다.

자세한 내용은 AWS Distro for OpenTelemetry와 X-Ray SDKs 중에서 선택 단원을 참조하십시오.

AWS OpenTelemetry .NET용 배포판

AWS Distro for OpenTelemetry .NET을 사용하면 애플리케이션을 한 번 계측하고 상관관계가 있는 지표 및 추적을 Amazon CloudWatch AWS X-Ray및 Amazon OpenSearch Service를 포함한 여러 AWS 모니터링 솔루션으로 전송할 수 있습니다. AWS Distro for OpenTelemetry와 함께 X-Ray를 사용하려면 X-Ray에서 사용할 수 있는 OpenTelemetry SDK와 X-Ray에서 사용할 수 있는 AWS Distro for OpenTelemetry Collector라는 두 가지 구성 요소가 필요합니다.

시작하려면 OpenTelemetry .NET용AWS 배포판 설명서를 참조하십시오.

AWS X-Ray 및 기타에서 AWS Distro for OpenTelemetry를 사용하는 방법에 대한 자세한 내용은 AWS Distro for OpenTelemetry 또는 AWS Distro for OpenTelemetry 설명서를 AWS 서비스참조하세요.

언어 지원 및 사용법에 대한 자세한 내용은 Github의AWS 관찰성을 참조하세요.

AWS X-Ray .NET용 SDK

.NET용 X-Ray SDK는 C# .NET 웹 애플리케이션, .NET Core 웹 애플리케이션 및 .NET Core 함수를 계측하기 위한 라이브러리입니다 AWS Lambda. 이 라이브러리는 트레이스 데이터를 생성하여 X-Ray 대몬(daemon)으로 전송하기 위한 클래스 및 메서드를 제공합니다. 여기에는 애플리케이션에서 제공하는 수신 요청과 애플리케이션이 다운스트림 AWS 서비스, HTTP 웹 APIs 및 SQL 데이터베이스에 수행하는 호출에 대한 정보가 포함됩니다.

> **Note**
>
> .NET용 X-Ray SDK는 오픈 소스 프로젝트입니다. 프로젝트를 따르고 GitHub(github.com/aws/aws-xray-sdk-dotnet)에서 문제 및 풀 요청을 제출할 수 있습니다.

웹 애플리케이션의 경우 메시지 핸들러를 웹 구성에 추가하여 수신 요청을 트레이스하는 것부터 시작합니다. 메시지 핸들러는 각 트레이스 요청에 대한 세그먼트를 생성하고, 응답이 전송되면 세그먼트를 완료합니다. 세그먼트가 열려 있는 동안에는 SDK 클라이언트의 메서드를 이용해 정보를 세그먼트에 추가하고 하위 세그먼트를 만들어 다운스트림 호출을 트레이스할 수 있습니다. 또한 SDK는 세그먼트가 열려 있는 동안 애플리케이션에서 발생하는 예외를 자동으로 기록합니다.

계측되는 애플리케이션 또는 서비스에 의해 호출되는 Lambda 함수의 경우, Lambda는 추적 헤더를 읽고 샘플링된 요청을 자동으로 추적합니다. 그 밖의 함수의 경우 수신 요청을 샘플링 및 추적하도록 Lambda를 구성합니다. 어느 경우든, Lambda는 세그먼트를 생성하여 X-Ray SDK에 제공합니다.

> **Note**
>
> Lambda의 X-Ray SDK는 선택 사항입니다. 이를 함수에 사용하지 않는 경우 여전히 서비스 맵에 Lambda 서비스용 노드와 각 Lambda 함수용 노트 하나가 포함됩니다. SDK를 추가하면 함수 코드를 계측하여 Lambda에 의해 기록되는 함수 세그먼트에 하위 세그먼트를 추가할 수 있습니다. 자세한 내용은 AWS Lambda 그리고 AWS X-Ray 섹션을 참조하세요.

그런 다음, X-Ray SDK for .NET를 사용하여 AWS SDK for .NET 클라이언트를 계측 합니다. 계측된 클라이언트를 사용하여 다운스트림 AWS 서비스 또는 리소스를 호출할 때마다 SDK는 하위 세그먼트에서 호출에 대한 정보를 기록합니다. AWS 서비스 및 서비스 내에서 액세스하는 리소스는 트레이스 맵에 다운스트림 노드로 표시되므로 개별 연결에서 오류 및 제한 문제를 식별하는 데 도움이 됩니다.

또한 .NET용 X-Ray SDK은 HTTP 웹 API 및 SQL 데이터베이스에 대한 다운스트림 호출의 계측도 제공합니다. `System.Net.HttpWebRequest`에 대한 `GetResponseTraced` 확장 메서드는 발신 HTTP 호출을 트레이스합니다. .NET용 X-Ray SDK 버전의 `SqlCommand`를 사용하여 SQL 쿼리를 계측할 수 있습니다.

SDK를 사용하기 시작한 후에, 레코더와 메시지 핸들러를 구성하여 SDK 동작을 구성하십시오. 플러그인을 추가해 애플리케이션을 실행하는 컴퓨팅 리소스에 대한 데이터를 기록하고, 샘플링 규칙을 정의해 샘플링 동작을 구성하고, 로그 레벨을 설정해 애플리케이션 로그의 SDK에서 표시되는 정보 수준을 조절할 수 있습니다.

요청에 대한 추가 정보와 애플리케이션이 주석 및 메타데이터에서 하는 작업을 기록합니다. 주석은 필터 표현식과 함께 사용할 수 있도록 인덱싱된 단순한 키 값 쌍이기 때문에, 특정 데이터를 포함한 트레이스를 검색할 수 있습니다. 메타데이터 항목은 제한이 적으며 JSON으로 직렬화할 수 있는 모든 객체와 어레이를 기록할 수 있습니다.

> **주석 및 메타데이터**
>
> 주석 및 메타데이터는 X SDK를 사용하여 세그먼트에 추가하는 임의의 텍스트입니다. 주석은 필터 표현식에서 사용하기 위해 인덱싱됩니다. 메타데이터는 인덱싱되지 않지만 X-Ray 콘솔 또는 API를 사용하여 원시 세그먼트에서 볼 수 있습니다. X-Ray에 대한 읽기 액세스가 부여된 사용자는 누구나 이 데이터를 볼 수 있습니다.

코드에 구성된 클라이언트가 많이 있다면, 구성된 클라이언트로 만든 각 요청의 하위 세그먼트를 대량으로 보관하는 요청 세그먼트 하나를 만들 수 있습니다. 사용자 지정 하위 세그먼트의 클라이언트 호출을 래핑해 하위 세그먼트를 조직하고 그룹화할 수 있습니다. 전체 함수나 특정 코드 부분에 대한 사용자 지정 하위 세그먼트를 만들고, 상위 세그먼트에 모든 것을 적는 대신 하위 세그먼트에 메타데이터와 주석을 기록할 수 있습니다.

SDK의 클래스 및 메서드에 대한 참조 문서는 다음을 참조하십시오.

- AWS X-Ray .NET용 SDK API 참조
- AWS X-Ray SDK for .NET Core API 참조

동일한 패키지에서 NET 및 .NET Core를 둘 다 지원하지만, 사용되는 클래스는 다릅니다. 이 장의 예제는 클래스가 .NET Core로 국한된 경우가 아니면 .NET API 참조로 연결됩니다.

요구 사항

.NET용 X-Ray SDK에는 .NET Framework 4.5 이상 및가 필요합니다 AWS SDK for .NET.

.NET Core 애플리케이션 및 함수의 경우, SDK에는 .NET Core 2.0 이상이 필요합니다.

애플리케이션에 .NET용 X-Ray SDK 추가하기

NuGet을 사용하여 애플리케이션에 .NET용 X-Ray SDK를 추가하세요.

Visual Studio의 NuGet 패키지 관리자를 사용하여 .NET용 X-Ray SDK를 설치하려면

1. [Tools], [NuGet Package Manager], [Manage NuGet Packages for Solution]을 선택합니다.

2. AWSXRayRecorder를 검색하세요.
3. 패키지를 선택하고 Install(설치)을 선택합니다.

종속성 관리

.NET용 X-Ray SDK는 Nuget에서 사용할 수 있습니다. 패키지 관리자를 사용하여 SDK를 설치합니다:

```
Install-Package AWSXRayRecorder -Version 2.10.1
```

`AWSXRayRecorder v2.10.1` nuget 패키지에는 다음과 같은 종속성을 가지고 있습니다:

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)
```

종속성 관리에 대한 자세한 내용은 Nuget 종속성 및 Nuget 종속성 해결에 대한 Microsoft 의 문서를 참조하십시오.

.NET용 X-Ray SDK 구성

애플리케이션이 실행되는 서비스에 대한 정보를 포함하거나, 기본 샘플링 동작을 수정하거나 요청에 적용되는 샘플링 규칙을 특정 경로에 추가하도록 플러그인을 사용하여 .NET용 X-Ray SDK를 구성할 수 있습니다.

.NET 웹 애플리케이션의 경우 키를 `Web.config` 파일의 `appSettings` 섹션에 추가합니다.

Example Web.config

```
<configuration>
  <appSettings>
```

```
    <add key="AWSXRayPlugins" value="EC2Plugin"/>
    <add key="SamplingRuleManifest" value="sampling-rules.json"/>
  </appSettings>
</configuration>
```

.NET Core의 경우, XRay를 최상위 키로 지정하여 appsettings.json 파일을 생성합니다.

Example .NET appsettings.json

```
{
  "XRay": {
    "AWSXRayPlugins": "EC2Plugin",
    "SamplingRuleManifest": "sampling-rules.json"
  }
}
```

그런 다음 애플리케이션 코드에서 구성 객체를 빌드하고 이 객체를 사용하여 X 레코더를 초기화합니다. 레코더를 초기화하기 전에 이 작업을 수행합니다.

Example .NET Core Program.cs – 레코더 구성

```
using Amazon.XRay.Recorder.Core;
...
AWSXRayRecorder.InitializeInstance(configuration);
```

.NET Core 웹 애플리케이션을 계측하는 경우 메시지 핸들러를 구성할 때 UseXRay 메서드에 구성 개체를 전달할 수도 있습니다. Lambda 함수의 경우 위와 같이 InitializeInstance 메서드를 사용합니다.

.NET Core 구성 API에 대한 자세한 내용은 docs.microsoft.com에서 ASP.NET Core 앱 구성을 참조하세요.

Sections

- 플러그인
- 샘플링 규칙
- 로깅 (.NET)
- 로깅(.NET Core)
- 환경 변수

플러그인

플러그인을 사용하여 애플리케이션을 호스팅하는 서비스에 대한 데이터를 추가합니다.

플러그인

- Amazon EC2 — `EC2Plugin`은 인스턴스 ID, 가용 영역 및 CloudWatch Logs 그룹을 추가합니다.
- Elastic Beanstalk – `ElasticBeanstalkPlugin`이 환경 이름, 버전 레이블 및 배포 ID를 추가합니다.
- Amazon ECS — `ECSPlugin`이 컨테이너 ID를 추가합니다.

플러그인을 사용하려면 `AWSXRayPlugins` 설정을 추가하여 .NET용 X-Ray SDK 클라이언트를 구성합니다. 여러 플러그인을 애플리케이션에 적용하는 경우 플러그인을 모두 동일 설정 안에 쉼표로 구분하여 지정합니다.

Example Web.config - 플러그인

```
<configuration>
  <appSettings>
    <add key="AWSXRayPlugins" value="EC2Plugin,ElasticBeanstalkPlugin"/>
  </appSettings>
</configuration>
```

Example .NET Core appsettings.json – 플러그인

```
{
  "XRay": {
    "AWSXRayPlugins": "EC2Plugin,ElasticBeanstalkPlugin"
  }
}
```

샘플링 규칙

SDK는 X-Ray 콘솔에서 정의하는 샘플링 규칙을 사용하여 기록할 요청을 결정합니다. 기본 규칙은 매초 첫 번째 요청을 추적하고, 모든 서비스에서 추가 요청의 5%를 X-Ray로 추적 전송합니다. X-Ray 콘솔에서 추가 규칙을 생성하여 각 애플리케이션에 대해 기록되는 데이터의 양을 사용자 지정합니다.

SDK는 사용자 지정 규칙을 정의된 순서대로 적용합니다. 요청이 여러 사용자 지정 규칙과 일치하는 경우 SDK는 첫 번째 규칙만 적용합니다.

> **Note**
>
> SDK가 샘플링 규칙을 가져오기 위해 X-Ray에 연결할 수 없는 경우, 매초 첫 번째 요청과 호스트당 추가 요청의 5%에 대한 기본 로컬 규칙으로 되돌아갑니다. 호스트가 샘플링 API를 직접 호출할 수 있는 권한이 없거나, X-Ray 대몬(daemon)에 연결할 수 없을 경우 이러한 상황이 발생할 수 있고 이 대몬(daemon)은 SDK에서 수행한 API 직접 호출에 대한 TCP 프록시 역할을 합니다.

JSON 문서에서 샘플링 규칙을 불러오도록 SDK를 구성할 수도 있습니다. SDK는 X-Ray 샘플링을 사용할 수 없을 경우 로컬 규칙을 백업으로 사용하거나, 로컬 규칙을 전용으로 사용할 수 있습니다.

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
  }
}
```

이 예에서는 하나의 사용자 지정 규칙과 기본 규칙을 정의합니다. 사용자 지정 규칙은 최소 추적 요청 수 없이 5% 샘플링 비율을 `/api/move/` 아래 경로에 적용합니다. 기본 규칙은 매초 최초 요청과 추가 요청의 10%를 추적합니다.

로컬로 규칙의 정의할 때의 단점은 고정 대상이 X-Ray 서비스를 통해 관리되는 대신, 레코더의 각 인스턴스별로 독립적으로 적용된다는 것입니다. 호스트를 많이 배포할수록 고정 속도가 크게 증대하기 때문에 기록되는 데이터의 양을 제어하기가 어려워집니다.

에서는 샘플링 속도를 수정할 수 AWS Lambda없습니다. 구성된 서비스가 함수를 호출하는 경우 해당 서비스에서 샘플링한 요청을 생성한 호출이 Lambda에 의해 기록됩니다. 활성 추적이 활성화되고 트레이스 헤더가 없는 경우 Lambda에서 샘플링 결정을 내립니다.

백업 규칙을 구성하려면 `SamplingRuleManifest` 설정을 사용하여 파일에서 샘플링 규칙을 로드하도록 .NET용 X-Ray SDK에 지시합니다.

Example .NET Web.config - 샘플링 규칙

```
<configuration>
  <appSettings>
    <add key="SamplingRuleManifest" value="sampling-rules.json"/>
  </appSettings>
</configuration>
```

Example .NET Core appsettings.json – 샘플링 규칙

```
{
  "XRay": {
    "SamplingRuleManifest": "sampling-rules.json"
  }
}
```

로컬 규칙만 사용하려면 `LocalizedSamplingStrategy`를 사용하여 레코더를 빌드합니다. 백업 규칙을 구성한 경우 해당 구성을 제거합니다.

Example .NET global.asax – 로컬 샘플링 규칙

```
var recorder = new AWSXRayRecorderBuilder().WithSamplingStrategy(new
 LocalizedSamplingStrategy("samplingrules.json")).Build();
AWSXRayRecorder.InitializeInstance(recorder: recorder);
```

Example .NET Core Program.cs – 로컬 샘플링 규칙

```
var recorder = new AWSXRayRecorderBuilder().WithSamplingStrategy(new
 LocalizedSamplingStrategy("sampling-rules.json")).Build();
AWSXRayRecorder.InitializeInstance(configuration,recorder);
```

로깅 (.NET)

.NET용 X-Ray SDK는 AWS SDK for .NET와 동일한 로깅 메커니즘을 사용합니다. AWS SDK for .NET 출력을 로깅하도록 애플리케이션을 이미 구성한 경우 .NET용 X-Ray SDK의 출력에도 동일한 구성이 적용됩니다.

로깅을 구성하려면 `aws`라는 구성 섹션을 `App.config` 파일이나 `Web.config` 파일에 추가합니다.

Example Web.config - 로깅

```
...
<configuration>
  <configSections>
    <section name="aws" type="Amazon.AWSSection, AWSSDK.Core"/>
  </configSections>
  <aws>
    <logging logTo="Log4Net"/>
  </aws>
</configuration>
```

자세한 내용은 AWS SDK for .NET 개발자 안내서의 AWS SDK for .NET 애플리케이션 구성을 참조하십시오.

로깅(.NET Core)

.NET용 X-Ray SDK는 AWS SDK for .NET와 동일한 로깅 옵션을 사용합니다. .NET Core 응용 프로그램의 로깅을 구성하려면 로깅 옵션을 `AWSXRayRecorder.RegisterLogger` 메서드에 전달하십시오.

예를 들어 log4net을 사용하려면 로거, 출력 형식 및 파일 위치를 정의하는 구성 파일을 생성합니다.

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>
```

그런 다음 로거를 생성하고 프로그램 코드에 해당 구성을 적용합니다.

Example .NET Core Program.cs — 로깅

```
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)
  {
  ...
  }
}
```

log4net 구성에 대한 자세한 내용은 logging.apache.org의 구성을 참조하십시오.

환경 변수

환경 변수를 사용하여 .NET용 X-Ray SDK를 구성할 수 있습니다. SDK는 다음 변수를 지원합니다.

- `AWS_XRAY_TRACING_NAME` – SDK가 세그먼트에 사용할 서비스 이름을 설정합니다. 서블릿 필터의 세그먼트 이름 지정 전략에 설정한 서비스 이름을 재정의합니다.
- `AWS_XRAY_DAEMON_ADDRESS` – X-Ray 대몬(daemon) 리스너의 호스트와 포트를 설정합니다. 기본적으로 SDK는 추적 데이터(UDP)와 샘플링(TCP) 모두에 `127.0.0.1:2000`을 사용합니다. 다른 포트에서 수신 대기하도록 대몬(daemon)을 구성한 경우 또는 다른 호스트에서 실행 중인 경우 이 변수를 사용합니다.

형식

- 동일한 포트 – *address*:*port*
- 다른 포트 – tcp:*address*:*port* udp:*address*:*port*

- AWS_XRAY_CONTEXT_MISSING – 열려 있는 세그먼트가 없는 경우 구성된 코드가 데이터를 기록하려고 할 때 발생하는 예외를 방지하려면 RUNTIME_ERROR로 설정합니다.

  유효한 값

  - RUNTIME_ERROR— 런타임 예외가 발생합니다.
  - LOG_ERROR – 오류를 기록하고 계속합니다 (기본값).
  - IGNORE_ERROR— 오류를 무시하고 계속합니다.

  열린 요청이 없을 때 실행되는 시작 코드 또는 새 스레드를 생성하는 코드에서 계측된 클라이언트를 사용하려고 하는 경우 누락된 세그먼트 또는 하위 세그먼트와 관련된 오류가 발생할 수 있습니다.

.NET용 X-Ray SDK로 수신 HTTP 요청 계측하기

X-Ray SDK를 사용하여 애플리케이션이 Amazon EC2 AWS Elastic Beanstalk또는 Amazon ECS의 EC2 인스턴스에서 처리하는 수신 HTTP 요청을 추적할 수 있습니다.

메시지 핸들러를 사용하여 수신 HTTP 요청을 구성합니다. X-Ray 메시지 핸들러를 애플리케이션에 추가할 때 .NET용 X-Ray SDK가 샘플링된 각 요청에 대해 세그먼트를 생성합니다. 이 세그먼트에는 HTTP 요청의 시간, 메서드 및 배치가 포함됩니다. 추가로 구성하면 이 세그먼트의 하위 세그먼트가 생성됩니다.

> **Note**
>
> AWS Lambda 함수의 경우 Lambda는 샘플링된 각 요청에 대해 세그먼트를 생성합니다. 자세한 내용은 AWS Lambda 그리고 AWS X-Ray 섹션을 참조하세요.

각 세그먼트에는 서비스 맵 안에서 애플리케이션을 식별하는 이름이 있습니다. 이 세그먼트의 이름이 정적으로 지정되도록 하거나, 수신 요청의 호스트 헤더를 기반으로 SDK가 동적으로 이름을 지정하도록 구성할 수 있습니다. 동적 이름 지정을 사용하면 요청의 도메인 이름에 따라 그룹을 추적하고 이름이 예상 패턴과 일치하지 않을 경우(예: 호스트 헤더가 위조된 경우) 기본 이름을 적용할 수 있습니다.

> **전달된 요청**
>
> 로드 밸런서 또는 기타 중개자가 애플리케이션으로 요청을 전달하는 경우, X-Ray는 IP 패킷 내 소스 IP가 아니라 요청의 `X-Forwarded-For` 헤더로부터 클라이언트 IP를 가져옵니다. 전달된 요청에 대해 기록된 클라어언트 IP는 위조될 수 있으므로 신뢰하면 안 됩니다.

메시지 핸들러는 다음 정보를 포함하는 `http` 블록을 이용해 각 수신 요청에 대한 세그먼트를 생성합니다.

- HTTP 메서드 – GET, POST, PUT, DELETE 등.
- 클라이언트 주소 – 요청을 전송한 클라이언트의 IP 주소.
- 응답 코드 – 완료된 요청의 HTTP 응답 코드.
- 시간 – 시작 시간(요청 수신) 및 종료 시간(응답 전송).
- 유저 에이전트 — 요청에서 가져온 `user-agent`입니다.
- 콘텐츠 길이 — 응답의 `content-length`입니다.

Sections

수신 HTTP 요청 구성(.NET)

애플리케이션에 의해 저장된 요청을 구성하려면 `RegisterXRay` 파일의 `Init` 메서드에서 `global.asax`를 호출합니다.

Example global.asax – 메시지 핸들러

```
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");
    }
  }
}
```

수신 HTTP 요청 구성(.NET Core)

애플리케이션에서 제공하는 요청을 계측하려면 스타트업 클래스의 `Configure` 메서드에서 다른 미들웨어보다 먼저 `UseXRay` 메서드를 직접 호출해야 합니다. 이상적으로는 X-Ray 미들웨어가 요청을 처리하는 첫 번째 미들웨어이고 파이프라인에서 응답을 처리하는 마지막 미들웨어가 되어야 합니다.

> **Note**
>
> .NET Core 2.0의 경우 애플리케이션에 `UseExceptionHandler` 메서드가 있는 경우 예외가 기록되도록 `UseExceptionHandler` 메서드 다음에 `UseXRay`를 직접 호출해야 합니다.

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 메서드는 구성 객체를 2차 인수로 사용할 수도 있습니다.

```
app.UseXRay("MyApp", configuration);
```

세그먼트 이름 지정 전략 구성

AWS X-Ray 는 서비스 이름을 사용하여 애플리케이션을 식별하고 애플리케이션이 사용하는 다른 애플리케이션, 데이터베이스, 외부 APIs 및 AWS 리소스와 구분합니다. X-Ray SDK는 수신 요청에 대한 세그먼트를 생성할 때 해당 세그먼트의 이름 필드에 애플리케이션의 서비스 이름을 기록합니다.

X-Ray SDK는 HTTP 요청 헤더의 호스트 이름 뒤에 세그먼트를 지정할 수 있습니다. 그러나 이 헤더가 위조되면 서비스 맵에 예기치 않은 노드가 발생할 수 있습니다. 위조된 호스트 헤더가 포함된 요청으로 인해 SDK가 잘못된 세그먼트 이름을 지정하는 현상을 방지하려면 들어오는 요청의 기본 이름을 지정해야 합니다.

애플리케이션이 여러 도메인의 요청을 처리하는 경우, 동적 이름 지정 전략을 사용하여 이를 세그먼트 이름에 반영하도록 SDK를 구성할 수 있습니다. 동적 이름 지정 전략을 사용하면 SDK가 예상 패턴과 일치하는 요청에 호스트 이름을 사용하고, 그렇지 않은 요청에 기본 이름을 적용할 수 있습니다.

예를 들어, 하나의 애플리케이션이 세 개의 하위 도메인 (`www.example.com`, `api.example.com`, `static.example.com`) 에 요청을 전송할 수 있습니다. `*.example.com` 패턴으로 동적 이름 지정 전략을 사용하여 각 하위도메인의 세그먼트를 서로 다른 이름으로 표시하면 서비스 맵에 서비스 노드가 세 개 생깁니다. 이 패턴에 맞지 않는 호스트 이름의 요청이 애플리케이션에 수신되면, 사용자가 지정한 대체 이름의 네 번째 노드가 서비스 맵에 표시됩니다.

모든 요청 세그먼트에 같은 이름을 사용하려면 이전 단원과 같이 메시지 핸들러를 초기화할 때 애플리케이션 이름을 지정합니다. 이렇게 하면 `FixedSegmentNamingStrategy`를 만들어 `RegisterXRay` 메서드에 전달하는 것과 같은 효과가 있습니다.

```
AWSXRayASPNET.RegisterXRay(this, new FixedSegmentNamingStrategy("MyApp"));
```

> **Note**
>
> 코드에 정의한 기본 서비스 이름을 `AWS_XRAY_TRACING_NAME` 환경 변수를 사용하여 재정의할 수 있습니다.

동적 이름 지정 전략은 호스트 이름이 일치해야 하는 패턴 및 HTTP 요청의 호스트 이름이 패턴과 일치하지 않는 경우 사용할 기본 이름을 정의합니다. 동적으로 세그먼트의 이름을 지정하려면 DynamicSegmentNamingStrategy를 생성하고 이를 RegisterXRay 메서드에 전달합니다.

```
AWSXRayASPNET.RegisterXRay(this, new DynamicSegmentNamingStrategy("MyApp",
 "*.example.com"));
```

.NET용 X-Ray AWS SDK를 사용하여 SDK 호출 추적

애플리케이션이를 호출 AWS 서비스 하여 데이터를 저장하거나, 대기열에 쓰거나, 알림을 보내면 .NET용 X-Ray SDK는 하위 세그먼트에서 다운스트림으로 호출을 추적합니다. 해당 서비스(예: Amazon S3 버킷 또는 Amazon SQS 대기열) 내에서 액세스하는 추적 AWS 서비스 및 리소스는 X-Ray 콘솔의 추적 맵에 다운스트림 노드로 표시됩니다.

모든 AWS SDK for .NET 클라이언트를 생성하기 RegisterXRayForAllServices 전에를 호출하여 계측할 수 있습니다.

Example SampleController.cs - DynamoDB 클라이언트 계측

```
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;
    });
```

전체 서비스가 아니라 일부 서비스에 대해서만 클라이언트를 구성하려면 `RegisterXRayForAllServices` 대신에 `RegisterXRay`를 호출합니다. 강조 표시된 텍스트를 서비스의 클라이언트 인터페이스 이름으로 바꿉니다.

```
AWSSDKHandler.RegisterXRay<IAmazonDynamoDB>()
```

모든 서비스에 대해, X-Ray 콘솔에서 호출된 API의 이름을 볼 수 있습니다. 서비스 하위 집합에 대해서는 X-Ray SDK가 세그먼트에 정보를 추가하여 서비스 맵에서 추가 세분화를 제공합니다.

예를 들어 계측된 DynamoDB 클라이언트에서 직접 호출을 생성하는 경우 SDK가 특정 테이블을 대상으로 한 직접 호출에 대해 테이블 이름을 세그먼트에 추가합니다. 콘솔에서, 각 테이블이 개별 노드로 서비스 맵에 표시되고, 특정 테이블을 대상으로 하지 않은 직접 호출에 대해 일반 DynamoDB 노드가 표시됩니다.

Example 항목을 저장하기 위한 DynamoDB 직접 호출에 대한 하위 세그먼트

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

명명된 리소스에 액세스할 때 다음 서비스를 호출할 경우 서비스 맵에 추가 노드가 생성됩니다. 특정 리소스를 대상으로 하지 않는 경우 서비스를 직접 호출하면 해당 서비스에 대한 일반 노드가 생성됩니다.

- Amazon DynamoDB – 테이블 이름
- Amazon Simple Storage Service –버킷 및 키 이름

- Amazon Simple Queue Service – 대기열 이름

.NET용 X-Ray SDK로 다운스트림 HTTP 웹 서비스에 대한 호출 추적하기

애플리케이션이 마이크로서비스 또는 공개 HTTP API를 호출할 때 `GetResponseTraced`에 대해 .NET용 X-Ray SDK의 `System.Net.HttpWebRequest` 확장 메서드를 사용하여 해당 호출을 계측하고 API를 다운스트림 서비스로 서비스 그래프에 추가할 수 있습니다.

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();
}
```

비동기식 호출의 경우 `GetAsyncResponseTraced`를 사용합니다.

```
request.GetAsyncResponseTraced();
```

system.net.http.httpclient를 사용하는 경우 `HttpClientXRayTracingHandler` 위임 핸들러를 사용하여 호출을 기록합니다.

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);
}
```

다운스트림 웹 API에 대한 직접 호출을 계측할 때 .NET용 X-Ray SDK가 HTTP 요청 및 응답에 대한 정보를 사용하여 하위 세그먼트를 기록합니다. X-Ray는 하위 세그먼트를 사용하여 API에 대해 추정된 세그먼트를 생성합니다.

Example 다운스트림 HTTP 호출에 대한 하위 세그먼트

```
{
  "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 다운스트림 HTTP 호출에 대한 추정된 세그먼트

```
{
  "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
}
```

.NET용 X-Ray SDK로 SQL 쿼리 추적하기

.NET용 X-Ray SDK는 `SqlCommand` 대신 사용할 수 있는 `System.Data.SqlClient.SqlCommand`용 래퍼 클래스 `TraceableSqlCommand`를 제공합니다. `TraceableSqlCommand` 클래스를 사용하여 SQL 명령을 초기화할 수 있습니다.

동기 및 비동기 메서드를 사용하여 SQL 쿼리 추적

다음 예제에서는 `TraceableSqlCommand`를 사용하여 동기식 및 비동기식으로 SQL Server 쿼리를 자동으로 추적하는 방법을 보여 줍니다.

Example **`Controller.cs`** - SQL 클라이언트 구성(동기)

```
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();
  }
}
```

`ExecuteReaderAsync` 메서드를 사용하여 비동기식으로 쿼리를 실행할 수 있습니다.

Example **`Controller.cs`** - SQL 클라이언트 구성(비동기)

```
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();
  }
}
```

SQL Server에 수행된 SQL 쿼리 수집

`SqlCommand.CommandText` 캡처를 SQL 쿼리에서 생성된 하위 세그먼트의 일부로 사용할 수 있습니다. `SqlCommand.CommandText`는 하위 세그먼트 JSON에서 `sanitized_query` 필드로 나타납니다. 보안을 위해 이 기능은 기본적으로 비활성화 상태입니다.

> **Note**
>
> 중요한 정보를 일반 텍스트로 SQL 쿼리에 포함시키는 경우 수집 기능을 활성화하지 마십시오.

다음 두 가지 방법으로 SQL 쿼리 수집을 활성화할 수 있습니다.

- 애플리케이션에 대한 글로벌 구성에서 `CollectSqlQueries` 속성을 `true`로 설정합니다.
- `TraceableSqlCommand` 인스턴스를 `collectSqlQueries` 파라미터를 `true`로 설정하여 인스턴스 내의 호출을 수집합니다.

글로벌 CollectSqlQueries 속성 활성화

다음 예제에서는 .NET 및 .NET Core에 대해 `CollectSqlQueries` 속성을 활성화하는 방법을 보여 줍니다.

.NET

.NET의 애플리케이션에 대한 글로벌 구성에서 `CollectSqlQueries` 속성을 `true`로 설정하려면 다음과 같이 `App.config` 또는 `Web.config` 파일의 `appsettings`을 수정합니다.

Example **`App.config`** 또는 **`Web.config`** – 글로벌로 SQL 쿼리 수집 활성화

```
<configuration>
<appSettings>
    <add key="CollectSqlQueries" value="true">
```

```
</appSettings>
</configuration>
```

.NET Core

.NET Core의 애플리케이션에 대한 글로벌 구성에서 `CollectSqlQueries` 속성을 `true`로 설정하려면 다음과 같이 X-Ray 키에 따라 `appsettings.json` 파일을 수정합니다.

Example **`appsettings.json`** – 글로벌로 SQL 쿼리 수집 활성화

```
{
  "XRay": {
    "CollectSqlQueries":"true"
  }
}
```

collectSqlQueries 파라미터 활성화

`TraceableSqlCommand` 인스턴스의 `collectSqlQueries` 파라미터를 `true`로 설정하여 해당 인스턴스에서 수행된 SQL Server 쿼리에 대한 SQL 쿼리 텍스트를 수집할 수 있습니다. 파라미터를 `false`로 설정하면 `TraceableSqlCommand` 인스턴스에 대한 `CollectSqlQuery` 기능이 비활성화됩니다.

> **Note**
>
> `TraceableSqlCommand` 인스턴스의 `collectSqlQueries` 값은 `CollectSqlQueries` 속성의 글로벌 구성에서 설정된 값을 재정의합니다.

Example 예제 **`Controller.cs`** – 인스턴스에 대한 SQL 쿼리 수집 활성화

```
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();
  }
}
```

추가 하위 세그먼트 생성

하위 세그먼트는 추적의 세그먼트를 확장하여 요청을 처리하기 위해 완료된 작업에 대한 세부 정보를 표시합니다. 계측되는 클라이언트에서 직접 호출할 때마다, X-Ray SDK는 하위 세그먼트 안에 생성된 정보를 기록합니다. 추가 하위 세그먼트를 생성하여 다른 하위 세그먼트를 그룹화하거나, 코드 섹션의 성능을 평가하거나, 주석 및 메타데이터를 기록할 수 있습니다.

하위 세그먼트를 관리하려면 `BeginSubsegment` 및 `EndSubsegment` 메서드를 사용합니다. `try` 블록에서 하위 세그먼트의 작업을 수행하고 `AddException`을 사용하여 예외를 트레이스합니다. `finally` 블록에서 `EndSubsegment`를 호출하여 하위 세그먼트가 닫히도록 합니다.

Example Controller.cs – 사용자 지정 하위 세그먼트

```
AWSXRayRecorder.Instance.BeginSubsegment("custom method");
try
{
  DoWork();
}
catch (Exception e)
{
  AWSXRayRecorder.Instance.AddException(e);
}
finally
{
  AWSXRayRecorder.Instance.EndSubsegment();
}
```

하위 세그먼트를 세그먼트 또는 다른 하위 세그먼트 내에서 생성할 경우 .NET용 X-Ray SDK가 해당 하위 세그먼트에 대해 ID를 생성하고 시작 시간 및 종료 시간을 기록합니다.

Example 메타데이터가 포함된 하위 세그먼트

```
"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"
    }
  },
```

.NET용 X-Ray SDK로 세그먼트에 주석 및 메타데이터 추가하기

주석 및 메타데이터와 함께 요청, 환경 또는 애플리케이션에 대한 추가 정보를 기록할 수 있습니다. X-Ray SDK에서 생성하는 세그먼트 또는 사용자가 생성하는 사용자 지정 하위 세그먼트에 주석 및 메타데이터를 추가할 수 있습니다.

주석은 문자열, 숫자 또는 부울 값과 결합한 키-값 페어입니다. 주석은 필터 표현식에서 사용하기 위해 인덱싱됩니다. 주석은 콘솔의 트레이스를 그룹화할 때 사용할 데이터를 기록하거나 `GetTraceSummaries` API를 직접 호출할 때 사용하세요.

메타데이터는 객체 및 목록을 포함한 모든 유형의 값을 가질 수 있는 키-값 페어지만, 필터 표현식에 사용할 수 있도록 인덱싱되지는 않습니다. 트레이스에 저장하고 싶지만 검색에는 사용하지 않을 추가 데이터는 메타데이터를 사용하여 기록하십시오.

Sections

- .NET용 X-Ray SDK로 주석 기록하기
- .NET용 X-Ray SDK로 메타데이터 기록하기

.NET용 X-Ray SDK로 주석 기록하기

주석을 사용하여 검색용으로 인덱싱할 정보를 세그먼트나 하위 세그먼트에 기록하십시오.

X-Ray의 모든 주석에는 다음이 필요합니다.

주석 요구 사항

- 키 - X-Ray 주석의 키는 최대 500자의 영숫자를 포함할 수 있습니다. 점이나 마침표(.) 이외의 공백이나 기호를 사용할 수 없습니다.
- 값 - X-Ray 주석의 값은 최대 1,000자의 유니코드 문자를 포함할 수 있습니다.
- 주석 수 - 트레이스당 최대 50개의 주석을 사용할 수 있습니다.

AWS Lambda 함수 외부의 주석을 기록하려면

1. AWSXRayRecorder의 인스턴스를 생성합니다.

```
using Amazon.XRay.Recorder.Core;
...
AWSXRayRecorder recorder = AWSXRayRecorder.Instance;
```

2. 문자열 키와 부울, Int32, Int64, Double 또는 문자열 값으로 addAnnotation을 호출합니다.

```
recorder.AddAnnotation("mykey", "my value");
```

다음 예에서는 점이 포함된 문자열 키와 부울, 숫자 또는 문자열 값을 사용하여 putAnnotation을 직접 호출하는 방법을 보여줍니다.

```
document.putAnnotation("testkey.test", "my value");
```

AWS Lambda 함수 내부의 주석을 기록하려면

Lambda 함수 내의 세그먼트와 하위 세그먼트는 모두 Lambda 런타임 환경에서 관리됩니다. Lambda 함수 내의 세그먼트 또는 하위 세그먼트에 주석을 추가하려면 다음을 수행해야 합니다.

1. Lambda 함수 내에 세그먼트 또는 하위 세그먼트를 생성합니다.
2. 세그먼트 또는 하위 세그먼트에 주석을 추가합니다.
3. 세그먼트 또는 하위 세그먼트를 종료합니다.

다음 코드 예제는 Lambda 함수 내의 하위 세그먼트에 주석을 추가하는 방법을 보여줍니다.

```
#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 SDK는 세그먼트 문서의 annotations 객체에 주석을 키-값 페어로 기록합니다. 동일한 키로 addAnnotation 작업을 두 번 직접 호출하면 동일한 세그먼트 또는 하위 세그먼트에 이전에 기록된 값을 덮어쓰게 됩니다.

특정 값을 포함한 주석이 있는 트레이스를 찾으려면 annotation[*key*] 키워드를 필터 표현식에 사용하십시오.

.NET용 X-Ray SDK로 메타데이터 기록하기

메타데이터를 사용하면 검색 내에 사용하기 위해 인덱싱할 필요가 없는 세그먼트 또는 하위 세그먼트에 대한 정보를 기록할 수 있습니다. 메타데이터 값은 문자열, 숫자, 부울 또는 JSON 객체나 어레이에 직렬화할 수 있는 다른 모든 객체가 될 수 있습니다.

메타데이터 기록 방법

1. 다음 코드 예시와 같이 AWSXRayRecorder의 인스턴스를 가져옵니다.

   ```
   using Amazon.XRay.Recorder.Core;
   ...
   AWSXRayRecorder recorder = AWSXRayRecorder.Instance;
   ```

2. 다음 코드 예시와 같이 문자열 네임스페이스, 문자열 키 및 객체 값을 사용하여 AddMetadata를 직접 호출합니다.

   ```
   recorder.AddMetadata("my namespace", "my key", "my value");
   ```

   다음 코드 예시와 같이 키와 값 페어만 사용하여 AddMetadata 작업을 직접 호출할 수도 있습니다.

   ```
   recorder.AddMetadata("my key", "my value");
   ```

네임스페이스 값을 지정하지 않으면 X-Ray SDK에서 `default`를 사용합니다. 동일한 키로 `AddMetadata` 작업을 두 번 직접 호출하면 동일한 세그먼트 또는 하위 세그먼트에 이전에 기록된 값을 덮어쓰게 됩니다.

Ruby로 작업

Ruby 애플리케이션을 계측하여 트레이스를 X-Ray로 전송하는 방법에는 두 가지가 있습니다.

- AWS Distro for OpenTelemetry Ruby - AWS Distro for OpenTelemetry Collector를 통해 상관관계가 있는 지표 및 트레이스를 Amazon CloudWatch AWS X-Ray및 Amazon OpenSearch Service를 포함한 여러 AWS 모니터링 솔루션으로 전송하기 위한 오픈 소스 라이브러리 세트를 제공하는 AWS 배포입니다.
- AWS X-Ray SDK for Ruby - X-Ray 데몬을 통해 트레이스를 생성하고 X-Ray로 전송하기 위한 라이브러리 세트입니다.

자세한 내용은 AWS Distro for OpenTelemetry와 X-Ray SDKs 중에서 선택 단원을 참조하십시오.

AWS OpenTelemetry Ruby용 배포판

AWS Distro for OpenTelemetry(ADOT) Ruby를 사용하면 애플리케이션을 한 번 계측하고 상관관계가 있는 지표 및 추적을 Amazon CloudWatch AWS X-Ray, Amazon OpenSearch Service를 포함한 여러 AWS 모니터링 솔루션으로 전송할 수 있습니다. ADOT와 함께 X-Ray를 사용하려면 두 가지 구성 요소가 필요합니다. X-Ray와 함께 사용할 수 있는 OpenTelemetry SDK와 X-Ray와 함께 사용할 수 있는 OpenTelemetry Collector를 위한AWS 배포판입니다.

시작하려면 Ruby용 OpenTelemetry AWS 배포판 설명서를 참조하십시오.

AWS X-Ray 및 기타에서 AWS Distro for OpenTelemetry를 사용하는 방법에 대한 자세한 내용은 AWS Distro for OpenTelemetry 또는 AWS Distro for OpenTelemetry 설명서를 AWS 서비스참조하세요.

언어 지원 및 사용법에 대한 자세한 내용은 Github의AWS 관찰성을 참조하세요.

AWS X-Ray SDK for Ruby

X-Ray SDK는 트레이스 데이터를 생성하고 X-Ray 대몬(daemon)에 보내기 위한 클래스 및 메서드를 제공하는 Ruby 웹 애플리케이션용 라이브러리 집합입니다. 트레이스 데이터에는 애플리케이션에서 제공하는 수신 HTTP 요청과 애플리케이션이 AWS SDK, HTTP 클라이언트 또는 활성 레코드 클라이언트를 사용하여 다운스트림 서비스에 수행하는 호출에 대한 정보가 포함됩니다. 또한 수동으로 세그먼트를 생성하고 디버그 정보를 주석 및 메타데이터에 추가할 수도 있습니다.

gemfile에 추가하고 `bundle install`를 실행하면 SDK를 다운로드할 수 있습니다.

Example Gemfile

```
gem 'aws-sdk'
```

Rails를 사용하는 경우 들어오는 요청을 추적하기 위해 먼저 X-Ray SDK 미들웨어를 추가하세요. 요청 필터는 세그먼트를 생성합니다. 세그먼트가 열려 있는 동안에는 SDK 클라이언트의 메서드를 이용해 정보를 세그먼트에 추가하고 하위 세그먼트를 만들어 다운스트림 호출을 트레이스할 수 있습니다. 또한 SDK는 세그먼트가 열려 있는 동안 애플리케이션에서 발생하는 예외를 자동으로 기록합니다. 다른 애플리케이션의 경우 세그먼트를 수동으로 만들 수 있습니다.

그런 다음 연결된 라이브러리를 패치하도록 레코더를 구성하여 X-Ray SDK를 사용하여 AWS SDK for Ruby, HTTP 및 SQL 클라이언트를 계측합니다. 계측된 클라이언트를 사용하여 다운스트림 AWS 서비스 또는 리소스를 호출할 때마다 SDK는 하위 세그먼트에 호출에 대한 정보를 기록합니다. 서비스 내에서 액세스하는 AWS 서비스 리소스는 트레이스 맵에 다운스트림 노드로 표시되므로 개별 연결에서 오류 및 제한 문제를 식별하는 데 도움이 됩니다.

SDK를 이용하게 되면 레코더를 구성하여 SDK 동작을 사용자 지정합니다. 플러그인을 추가해 애플리케이션을 실행하는 컴퓨팅 리소스에 대한 데이터를 기록하고, 샘플링 규칙을 정의해 샘플링 동작을 구성하고, 로거를 제공하여 애플리케이션 로그의 SDK에서 표시되는 정보 수준을 조절할 수 있습니다.

요청에 대한 추가 정보와 애플리케이션이 주석 및 메타데이터에서 하는 작업을 기록합니다. 주석은 필터 표현식과 함께 사용할 수 있도록 인덱싱된 단순한 키 값 쌍이기 때문에, 특정 데이터를 포함한 트레이스를 검색할 수 있습니다. 메타데이터 항목은 제한이 적으며 JSON으로 직렬화할 수 있는 모든 객체와 어레이를 기록할 수 있습니다.

> **주석 및 메타데이터**
>
> 주석 및 메타데이터는 X SDK를 사용하여 세그먼트에 추가하는 임의의 텍스트입니다. 주석은 필터 표현식에서 사용하기 위해 인덱싱됩니다. 메타데이터는 인덱싱되지 않지만 X-Ray 콘솔 또는 API를 사용하여 원시 세그먼트에서 볼 수 있습니다. X-Ray에 대한 읽기 액세스가 부여된 사용자는 누구나 이 데이터를 볼 수 있습니다.

코드에 구성된 클라이언트가 많이 있다면, 구성된 클라이언트로 만든 각 직접 호출의 하위 세그먼트를 대량으로 보관하는 요청 세그먼트 하나를 만들 수 있습니다. 사용자 지정 하위 세그먼트의 클라이언트 호출을 래핑해 하위 세그먼트를 조직하고 그룹화할 수 있습니다. 전체 함수나 특정 코드 부분에 대한 사용자 지정 하위 세그먼트를 만들고, 상위 세그먼트에 모든 것을 적는 대신 하위 세그먼트에 메타데이터와 주석을 기록할 수 있습니다.

SDK의 클래스 및 메서드에 대한 참조 문서는 AWS X-Ray SDK for Ruby API Reference를 참조하십시오.

요구 사항

X-Ray SDK는 Ruby 2.3 이상이 필요하며 다음 라이브러리와 호환됩니다:

- AWS SDK for Ruby 버전 3.0 이상
- Rails 버전 5.1 이상

Ruby용 X-Ray SDK 구성

Ruby용 X-Ray SDK에는 전역 레코더를 제공하는 `XRay.recorder`라는 클래스가 있습니다. 수신 HTTP 호출에 대해 세그먼트를 생성하는 미들웨어를 사용자 지정하도록 전역 레코더를 구성할 수 있습니다.

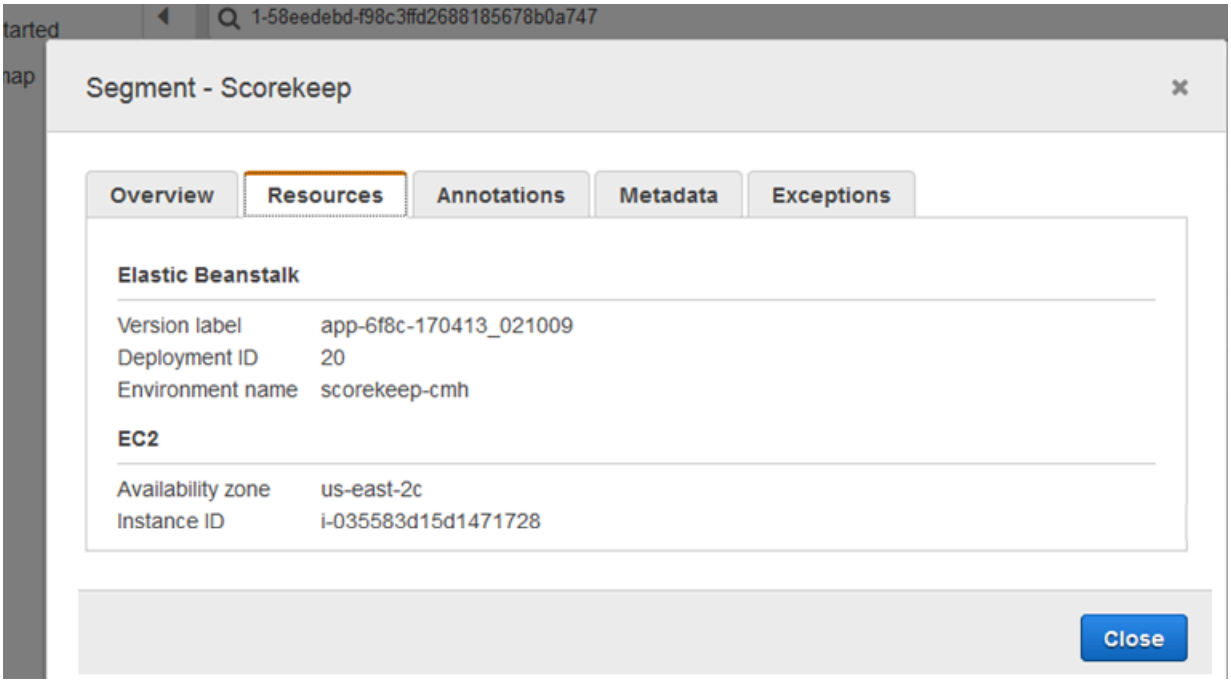
Sections

서비스 플러그인

`plugins`을 사용하여 애플리케이션을 호스팅하는 서비스에 대한 정보를 기록할 수 있습니다.

플러그인

- Amazon EC2 – `ec2`이 인스턴스 ID와 가용 영역을 추가합니다.
- Elastic Beanstalk – `elastic_beanstalk`이 환경 이름, 버전 레이블 및 배포 ID를 추가합니다.
- Amazon ECS — `ecs`이 컨테이너 ID를 추가합니다.



플러그인을 사용하려면 레코더로 전달하는 구성 객체에서 해당 플러그인을 지정합니다.

Example main.rb – 플러그인 구성

```
my_plugins = %I[ec2 elastic_beanstalk]

config = {
  plugins: my_plugins,
  name: 'my app',
}

XRay.recorder.configure(config)
```

또한 코드에 설정된 값보다 우선하는 환경 변수를 사용하여 레코더를 구성할 수 있습니다.

SDK는 플러그인 설정을 사용하여 세그먼트에 `origin` 필드를 설정하기도 합니다. 이는 애플리케이션을 실행하는 AWS 리소스 유형을 나타냅니다. 여러 플러그인을 사용하는 경우 SDK는 ElasticBeanstalk> EKS> ECS> EC2 순서로 확인하여 오리진을 결정합니다.

샘플링 규칙

SDK는 X-Ray 콘솔에서 정의하는 샘플링 규칙을 사용하여 기록할 요청을 결정합니다. 기본 규칙은 매초 첫 번째 요청을 추적하고, 모든 서비스에서 추가 요청의 5%를 X-Ray로 추적 전송합니다. X-Ray 콘솔에서 추가 규칙을 생성하여 각 애플리케이션에 대해 기록되는 데이터의 양을 사용자 지정합니다.

SDK는 사용자 지정 규칙을 정의된 순서대로 적용합니다. 요청이 여러 사용자 지정 규칙과 일치하는 경우 SDK는 첫 번째 규칙만 적용합니다.

> **Note**
>
> SDK가 샘플링 규칙을 가져오기 위해 X-Ray에 연결할 수 없는 경우, 매초 첫 번째 요청과 호스트당 추가 요청의 5%에 대한 기본 로컬 규칙으로 되돌아갑니다. 호스트가 샘플링 API를 직접 호출할 수 있는 권한이 없거나, X-Ray 대몬(daemon)에 연결할 수 없을 경우 이러한 상황이 발생할 수 있고 이 대몬(daemon)은 SDK에서 수행한 API 직접 호출에 대한 TCP 프록시 역할을 합니다.

JSON 문서에서 샘플링 규칙을 불러오도록 SDK를 구성할 수도 있습니다. SDK는 X-Ray 샘플링을 사용할 수 없을 경우 로컬 규칙을 백업으로 사용하거나, 로컬 규칙을 전용으로 사용할 수 있습니다.

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
  }
}
```

이 예에서는 하나의 사용자 지정 규칙과 기본 규칙을 정의합니다. 사용자 지정 규칙은 최소 추적 요청 수 없이 5% 샘플링 비율을 `/api/move/` 아래 경로에 적용합니다. 기본 규칙은 매초 최초 요청과 추가 요청의 10%를 추적합니다.

로컬로 규칙의 정의할 때의 단점은 고정 대상이 X-Ray 서비스를 통해 관리되는 대신, 레코더의 각 인스턴스별로 독립적으로 적용된다는 것입니다. 호스트를 많이 배포할수록 고정 속도가 크게 증대하기 때문에 기록되는 데이터의 양을 제어하기가 어려워집니다.

백업 규칙을 구성하려면 레코더로 전달하는 구성 객체에서 문서에 대한 해시를 정의합니다.

Example main.rb – 백업 규칙 구성

```
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)
```

샘플링 규칙을 독립적으로 저장하려면 별도 파일에 해시를 정의하고 파일이 애플리케이션으로 해시를 가져오도록 합니다.

Example config/sampling-rules.rb

```
my_sampling_rules =  {
  version: 1,
  default: {
    fixed_target: 1,
    rate: 0.1
  }
}
```

Example main.rb – 파일의 샘플링 규칙

```
require 'aws-xray-sdk'
require 'config/sampling-rules.rb'

config = {
```

```
  sampling_rules: my_sampling_rules,
  name: 'my app',
}
XRay.recorder.configure(config)
```

로컬 규칙만 사용하려면 샘플링 규칙을 요청하고 `LocalSampler`를 구성합니다.

Example main.rb – 로컬 샘플링 규칙

```
require 'aws-xray-sdk'
require 'aws-xray-sdk/sampling/local/sampler'

config = {
  sampler: LocalSampler.new,
  name: 'my app',
}
XRay.recorder.configure(config)
```

또한 샘플링을 비활성하하고 모든 수신 요청을 구성하도록 전역 레코더를 구성할 수 있습니다.

Example main.rb – 샘플링 비활성화

```
require 'aws-xray-sdk'
config = {
  sampling: false,
  name: 'my app',
}
XRay.recorder.configure(config)
```

로깅

기본적으로 레코더는 정보 레벨 이벤트를 `$stdout`로 출력합니다. 레코더로 전달하는 구성 객체에서 로거를 정의하여 로깅을 사용자 정의할 수 있습니다.

Example main.rb – 로깅

```
require 'aws-xray-sdk'
config = {
  logger: my_logger,
  name: 'my app',
}
```

```
XRay.recorder.configure(config)
```

디버그 로그를 사용하여 수동으로 하위 세그먼트를 생성할 때 미완료 하위 세그먼트와 같은 문제를 식별할 수 있습니다.

코드의 레코더 구성

`XRay.recorder`에 대한 `configure` 메서드에서 추가 설정을 사용할 수 있습니다.

- `context_missing` – 열려 있는 세그먼트가 없는 경우 구성된 코드가 데이터를 기록하려고 할 때 발생하는 예외를 방지하려면 `LOG_ERROR`로 설정합니다.
- `daemon_address` – X-Ray 대몬(daemon) 리스너의 호스트와 포트를 설정합니다.
- `name` – SDK가 세그먼트에 사용하는 서비스 이름을 설정합니다.
- `naming_pattern` – 동적 이름 지정을 사용하려면 도메인 이름 패턴을 설정합니다.
- `plugins` – 플러그인을 사용하여 애플리케이션의 AWS 리소스에 대한 정보를 기록합니다.
- `sampling` – 샘플링을 비활성화하려면 `false`로 설정합니다.
- `sampling_rules` – 샘플링 규칙이 포함된 해시를 설정합니다.

Example main.rb – 컨텍스트 누락 예외 비활성화

```
require 'aws-xray-sdk'
config = {
  context_missing: 'LOG_ERROR'
}

XRay.recorder.configure(config)
```

Rails를 사용한 레코더 구성

Rails 프레임워크를 사용하는 경우 `app_root/initializers` 아래의 Ruby 파일을 사용하여 전역 레코더의 옵션을 구성할 수 있습니다. X-Ray SDK는 Rails와 함께 사용할 수 있는 추가 구성 키를 지원합니다.

- `active_record` – Active Record 데이터베이스 트랜잭션에 대한 하위 세그먼트를 기록하려면 `true`로 설정합니다.

이름이 `Rails.application.config.xray`인 구성 객체에서 사용 가능한 설정을 구성합니다.

Example config/initializers/aws_xray.rb

```
Rails.application.config.xray = {
  name: 'my app',
  patch: %I[net_http aws_sdk],
  active_record: true
}
```

환경 변수

환경 변수를 사용하여 Ruby용 X-Ray SDK를 구성할 수 있습니다. SDK는 다음 변수를 지원합니다:

- `AWS_XRAY_TRACING_NAME` – SDK가 세그먼트에 사용할 서비스 이름을 설정합니다. 서블릿 필터의 세그먼트 이름 지정 전략에 설정한 서비스 이름을 재정의합니다.
- `AWS_XRAY_DAEMON_ADDRESS` – X-Ray 대몬(daemon) 리스너의 호스트와 포트를 설정합니다. 기본적으로 SDK는 `127.0.0.1:2000`에 트레이스 데이터를 보냅니다. 다른 포트에서 수신 대기하도록 대몬(daemon)을 구성한 경우 또는 다른 호스트에서 실행 중인 경우 이 변수를 사용합니다.
- `AWS_XRAY_CONTEXT_MISSING` – 열려 있는 세그먼트가 없는 경우 구성된 코드가 데이터를 기록하려고 할 때 발생하는 예외를 방지하려면 `RUNTIME_ERROR`로 설정합니다.

  유효한 값

  - `RUNTIME_ERROR`— 런타임 예외가 발생합니다.
  - `LOG_ERROR` – 오류를 기록하고 계속합니다 (기본값).
  - `IGNORE_ERROR`— 오류를 무시하고 계속합니다.

  열린 요청이 없을 때 실행되는 시작 코드 또는 새 스레드를 생성하는 코드에서 계측된 클라이언트를 사용하려고 하는 경우 누락된 세그먼트 또는 하위 세그먼트와 관련된 오류가 발생할 수 있습니다.

환경 변수는 코드에 설정된 값을 재정의합니다.

Ruby 미들웨어용 X-Ray SDK로 수신 요청 추적하기

X-Ray SDK를 사용하여 애플리케이션이 Amazon EC2 AWS Elastic Beanstalk또는 Amazon ECS의 EC2 인스턴스에서 처리하는 수신 HTTP 요청을 추적할 수 있습니다.

Rails를 사용하는 경우 Rails 미들웨어를 사용하여 수신 HTTP 요청을 구성합니다. 미들웨어를 애플리케이션에 추가하고 세그먼트 이름을 구성하면 Ruby용 X-Ray SDK는 샘플링된 각 요청에 대해 세그먼

트를 생성합니다. 추가 구성에 의해 생성된 모든 세그먼트는 HTTP 요청 및 응답에 대한 정보를 제공하는 요청 레벨 세그먼트의 하위 세그먼트가 됩니다. 이 정보에는 요청의 시간, 메서드 및 배치가 포함됩니다.

각 세그먼트에는 서비스 맵 안에서 애플리케이션을 식별하는 이름이 있습니다. 이 세그먼트의 이름이 정적으로 지정되도록 하거나, 수신 요청의 호스트 헤더를 기반으로 SDK가 동적으로 이름을 지정하도록 구성할 수 있습니다. 동적 이름 지정을 사용하면 요청의 도메인 이름에 따라 그룹을 추적하고 이름이 예상 패턴과 일치하지 않을 경우(예: 호스트 헤더가 위조된 경우) 기본 이름을 적용할 수 있습니다.

> **전달된 요청**
>
> 로드 밸런서 또는 기타 중개자가 애플리케이션으로 요청을 전달하는 경우, X-Ray는 IP 패킷 내 소스 IP가 아니라 요청의 `X-Forwarded-For` 헤더로부터 클라이언트 IP를 가져옵니다. 전달된 요청에 대해 기록된 클라이언트 IP는 위조될 수 있으므로 신뢰하면 안 됩니다.

요청이 전달되면 SDK는 세그먼트에 추가 필드를 설정하여 이를 나타냅니다. 세그먼트에 `x_forwarded_for`로 설정된 `true` 필드가 포함된 경우 클라이언트 IP는 HTTP 요청의 `X-Forwarded-For` 헤더로부터 가져옵니다.

미들웨어는 다음 정보를 포함하는 `http` 블록으로 각 수신 요청에 대한 세그먼트를 생성합니다.

- HTTP 메서드 – GET, POST, PUT, DELETE 등.
- 클라이언트 주소 – 요청을 전송한 클라이언트의 IP 주소.
- 응답 코드 – 완료된 요청의 HTTP 응답 코드.
- 시간 – 시작 시간(요청 수신) 및 종료 시간(응답 전송).
- 유저 에이전트 — 요청에서 가져온 `user-agent`입니다.
- 콘텐츠 길이 — 응답의 `content-length`입니다.

rails 미들웨어 사용

미들웨어를 사용하려면 필요한 railtie를 포함하도록 gemfile을 업데이트합니다.

Example Gemfile - rails

```
gem 'aws-xray-sdk', require: ['aws-xray-sdk/facets/rails/railtie']
```

또한 미들웨어를 사용하려면 트레이스 맵에서 애플리케이션을 나타내는 이름을 사용하여 레코더를 구성해야 합니다.

Example config/initializers/aws_xray.rb

```
Rails.application.config.xray = {
  name: 'my app'
}
```

수동으로 코드 구성

Rails를 사용하지 않는 경우 세그먼트를 수동으로 생성합니다. 수신되는 각 요청에 대해 세그먼트를 생성하거나 패치된 HTTP 또는 AWS SDK 클라이언트 주위에 세그먼트를 생성하여 레코더가 하위 세그먼트를 추가할 컨텍스트를 제공할 수 있습니다.

```
# 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
```

세그먼트 이름 지정 전략 구성

AWS X-Ray 는 서비스 이름을 사용하여 애플리케이션을 식별하고 애플리케이션이 사용하는 다른 애플리케이션, 데이터베이스, 외부 APIs 및 AWS 리소스와 구분합니다. X-Ray SDK는 수신 요청에 대한 세그먼트를 생성할 때 해당 세그먼트의 이름 필드에 애플리케이션의 서비스 이름을 기록합니다.

X-Ray SDK는 HTTP 요청 헤더의 호스트 이름 뒤에 세그먼트를 지정할 수 있습니다. 그러나 이 헤더가 위조되면 서비스 맵에 예기치 않은 노드가 발생할 수 있습니다. 위조된 호스트 헤더가 포함된 요청으로 인해 SDK가 잘못된 세그먼트 이름을 지정하는 현상을 방지하려면 들어오는 요청의 기본 이름을 지정해야 합니다.

애플리케이션이 여러 도메인의 요청을 처리하는 경우, 동적 이름 지정 전략을 사용하여 이를 세그먼트 이름에 반영하도록 SDK를 구성할 수 있습니다. 동적 이름 지정 전략을 사용하면 SDK가 예상 패턴과 일치하는 요청에 호스트 이름을 사용하고, 그렇지 않은 요청에 기본 이름을 적용할 수 있습니다.

예를 들어, 하나의 애플리케이션이 세 개의 하위 도메인 (`www.example.com`, `api.example.com`, `static.example.com`) 에 요청을 전송할 수 있습니다. `*.example.com` 패턴으로 동적 이름 지정 전략을 사용하여 각 하위도메인의 세그먼트를 서로 다른 이름으로 표시하면 서비스 맵에 서비스 노드가 세 개 생깁니다. 이 패턴에 맞지 않는 호스트 이름의 요청이 애플리케이션에 수신되면, 사용자가 지정한 대체 이름의 네 번째 노드가 서비스 맵에 표시됩니다.

모든 요청 세그먼트에 같은 이름을 사용하려면 이전 섹션에 나온 것처럼 레코더를 구성할 때 애플리케이션 이름을 지정합니다.

동적 이름 지정 전략은 호스트 이름이 일치해야 하는 패턴 및 HTTP 요청의 호스트 이름이 패턴과 일치하지 않는 경우 사용할 기본 이름을 정의합니다. 세그먼트 이름을 동적으로 지정하려면 config 해시에서 이름 지정 패턴을 지정합니다.

Example main.rb – 동적 이름 지정

```
config = {
  naming_pattern: '*mydomain*',
  name: 'my app',
}

XRay.recorder.configure(config)
```

패턴에서 '*'를 사용하여 문자열을 일치시키거나 '?'를 사용하여 단일 문자를 일치시킬 수 있습니다.

> **Note**
>
> 코드에 정의한 기본 서비스 이름을 `AWS_XRAY_TRACING_NAME` 환경 변수를 사용하여 재정의할 수 있습니다.

다운스트림 호출 구성을 위해 라이브러리 패치

다운스트림 호출을 구성하려면 Ruby용 X-Ray SDK를 사용하여 애플리케이션이 사용하는 라이브러리를 패치합니다. Ruby용 X-Ray SDK는 다음 라이브러리를 패치할 수 있습니다.

지원되는 라이브러리

- `net/http` – HTTP 클라이언트를 구성합니다.
- `aws-sdk` - AWS SDK for Ruby 클라이언트를 계측합니다.

패치된 라이브러리를 사용하면 Ruby용 X-Ray SDK는 직접 호출에 대한 하위 세그먼트를 생성하고 요청 및 응답의 정보를 기록합니다. SDK가 SDK 미들웨어 또는 `XRay.recorder.begin_segment` 호출에서 하위 세그먼트를 생성하려면 세그먼트를 사용할 수 있어야 합니다.

라이브러리를 패치하려면 X-Ray 레코더에 전달하는 구성 객체에서 해당 라이브러리를 지정합니다.

Example main.rb – 라이브러리 패치

```
require 'aws-xray-sdk'

config = {
  name: 'my app',
  patch: %I[net_http aws_sdk]
}

XRay.recorder.configure(config)
```

Ruby용 X-Ray AWS SDK를 사용하여 SDK 호출 추적

애플리케이션이를 호출 AWS 서비스 하여 데이터를 저장하거나, 대기열에 쓰거나, 알림을 보내면 Ruby용 X-Ray SDK는 하위 세그먼트에서 다운스트림으로 호출을 추적합니다. 해당 서비스(예: Amazon S3 버킷 또는 Amazon SQS 대기열) 내에서 액세스하는 추적 AWS 서비스 및 리소스는 X-Ray 콘솔의 추적 맵에 다운스트림 노드로 표시됩니다.

Ruby용 X-Ray SDK는 라이브러리를 패치할 때 모든 AWS SDK 클라이언트를 자동으로 계측합니다. `aws-sdk` 개별 클라이언트를 구성할 수는 없습니다.

모든 서비스의 경우, X-Ray 콘솔에서 호출된 API의 이름을 볼 수 있습니다. 서비스 하위 집합에 대해서는 X-Ray SDK가 세그먼트에 정보를 추가하여 서비스 맵에서 추가 세분화를 제공합니다.

예를 들어 계측된 DynamoDB 클라이언트에서 직접 호출을 생성하는 경우 SDK가 특정 테이블을 대상으로 한 직접 호출에 대해 테이블 이름을 세그먼트에 추가합니다. 콘솔에서, 각 테이블이 개별 노드로 서비스 맵에 표시되고, 특정 테이블을 대상으로 하지 않은 직접 호출에 대해 일반 DynamoDB 노드가 표시됩니다.

Example 항목을 저장하기 위한 DynamoDB 직접 호출에 대한 하위 세그먼트

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

명명된 리소스에 액세스할 때 다음 서비스를 호출할 경우 서비스 맵에 추가 노드가 생성됩니다. 특정 리소스를 대상으로 하지 않는 경우 서비스를 직접 호출하면 해당 서비스에 대한 일반 노드가 생성됩니다.

- Amazon DynamoDB – 테이블 이름
- Amazon Simple Storage Service –버킷 및 키 이름
- Amazon Simple Queue Service – 대기열 이름

X-Ray SDK를 사용하여 사용자 지정 하위 세그먼트 생성

하위 세그먼트는 추적의 세그먼트를 확장하여 요청을 처리하기 위해 완료된 작업에 대한 세부 정보를 표시합니다. 계측되는 클라이언트에서 직접 호출할 때마다, X-Ray SDK는 하위 세그먼트 안에 생성된 정보를 기록합니다. 추가 하위 세그먼트를 생성하여 다른 하위 세그먼트를 그룹화하거나, 코드 섹션의 성능을 평가하거나, 주석 및 메타데이터를 기록할 수 있습니다.

하위 세그먼트를 관리하려면 begin_subsegment 및 end_subsegment 메서드를 사용합니다.

```
subsegment = XRay.recorder.begin_subsegment name: 'annotations', namespace: 'remote'
my_annotations = { id: 12345 }
subsegment.annotations.update my_annotations
XRay.recorder.end_subsegment
```

함수에 대한 하위 세그먼트를 생성하려면 XRay.recorder.capture 호출에 함수를 래핑합니다.

```
XRay.recorder.capture('name_for_subsegment') do |subsegment|
  resp = myfunc() # myfunc is your function
  subsegment.annotations.update k1: 'v1'
  resp
end
```

하위 세그먼트를 세그먼트 또는 다른 하위 세그먼트 내에서 생성하면 X-Ray SDK가 해당 하위 세그먼트에 대해 ID를 생성하고 시작 시간 및 종료 시간을 기록합니다.

Example 메타데이터가 포함된 하위 세그먼트

```
"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"
    }
  },
```

Ruby용 X-Ray SDK로 세그먼트에 주석 및 메타데이터 추가하기

주석 및 메타데이터와 함께 요청, 환경 또는 애플리케이션에 대한 추가 정보를 기록할 수 있습니다. X-Ray SDK에서 생성하는 세그먼트 또는 사용자가 생성하는 사용자 지정 하위 세그먼트에 주석 및 메타데이터를 추가할 수 있습니다.

주석은 문자열, 숫자 또는 부울 값과 결합한 키-값 페어입니다. 주석은 필터 표현식에서 사용하기 위해 인덱싱됩니다. 주석은 콘솔의 트레이스를 그룹화할 때 사용할 데이터를 기록하거나 GetTraceSummaries API를 직접 호출할 때 사용하세요.

메타데이터는 객체 및 목록을 포함한 모든 유형의 값을 가질 수 있는 키-값 페어지만, 필터 표현식에 사용할 수 있도록 인덱싱되지는 않습니다. 트레이스에 저장하고 싶지만 검색에는 사용하지 않을 추가 데이터는 메타데이터를 사용하여 기록하십시오.

세그먼트에는 주석과 메타데이터 외에 사용자 ID 문자열도 기록할 수 있습니다. 사용자 ID는 세그먼트의 별도 필드에 기록되면 검색용으로 인덱스되지 않습니다.

Sections

Ruby용 X-Ray SDK로 주석 기록하기

주석을 사용하여 검색용으로 인덱싱할 정보를 세그먼트나 하위 세그먼트에 기록하십시오.

주석 요구 사항

- 키 - X-Ray 주석의 키는 최대 500자의 영숫자를 포함할 수 있습니다. 점이나 마침표(.) 이외의 공백이나 기호를 사용할 수 없습니다.
- 값 - X-Ray 주석의 값은 최대 1,000자의 유니코드 문자를 포함할 수 있습니다.
- 주석 수 - 트레이스당 최대 50개의 주석을 사용할 수 있습니다.

주석 기록 방법

1. `xray_recorder`에서 현재 세그먼트나 하위 세그먼트의 참조를 가져오십시오.

```
require 'aws-xray-sdk'
...
document = XRay.recorder.current_segment
```

or

```
require 'aws-xray-sdk'
...
document = XRay.recorder.current_subsegment
```

2. 해시 값을 사용해 `update`를 호출합니다.

```
my_annotations = { id: 12345 }
document.annotations.update my_annotations
```

다음은 점이 포함된 주석 키를 사용하여 update를 직접 호출하는 방법을 보여주는 예제입니다.

```
my_annotations = { testkey.test: 12345 }
document.annotations.update my_annotations
```

SDK는 세그먼트 문서의 annotations 객체에 주석을 키-값 페어로 기록합니다. 같은 키로 add_annotations을 두 번 직접 호출하면 같은 세그먼트나 하위 세그먼트에 기록했던 값을 덮어씁니다.

특정 값을 포함한 주석이 있는 트레이스를 찾으려면 annotation[*key*] 키워드를 필터 표현식에 사용하십시오.

Ruby용 X-Ray SDK로 메타데이터 기록하기

메타데이터를 이용해 검색용으로 인덱싱하지 않아도 되는 정보를 세그먼트나 하위 세그먼트에 기록하십시오. 메타데이터 값은 문자열, 숫자, 부울 또는 JSON 객체나 어레이에 직렬화할 수 있는 모든 객체가 될 수 있습니다.

메타데이터 기록 방법

1. xray_recorder에서 현재 세그먼트나 하위 세그먼트의 참조를 가져오십시오.

   ```
   require 'aws-xray-sdk'
   ...
   document = XRay.recorder.current_segment
   ```

   or

   ```
   require 'aws-xray-sdk'
   ...
   document = XRay.recorder.current_subsegment
   ```

2. 문자열 키, 부울, 숫자, 문자열 또는 객체 값 및 문자열 네임스페이스와 함께 metadata를 직접 호출합니다.

```
my_metadata = {
  my_namespace: {
    key: 'value'
  }
}
subsegment.metadata my_metadata
```

같은 키로 metadata을 두 번 직접 호출하면 같은 세그먼트나 하위 세그먼트에 기록했던 값을 덮어씁니다.

Ruby용 X-Ray SDK로 사용자 ID 기록하기

사용자 ID를 요청 세그먼트에 기록하여 요청을 보낸 사용자를 식별합니다.

사용자 ID 기록 방법

1. xray_recorder에서 현재 세그먼트에 대한 참조를 가져옵니다.

```
require 'aws-xray-sdk'
...
document = XRay.recorder.current_segment
```

2. 세그먼트의 사용자 필드를 요청을 보낸 사용자의 문자열 ID로 설정합니다.

```
segment.user = 'U12345'
```

컨트롤러에서 사용자를 설정하면 애플리케이션이 요청을 처리하는 순간부터 사용자 ID를 기록할 수 있습니다.

사용자 ID의 트레이스를 찾으려면, user 키워드를 필터 표현식에 적용하십시오.

X-Ray 계측에서 OpenTelemetry 계측으로 마이그레이션

X-Ray는 애플리케이션 추적 및 관찰성을 위한 기본 계측 표준으로 OpenTelemetry(OTel)로 전환하고 있습니다. 이 전략적 전환은 AWS 업계 모범 사례에 부합하며 고객에게 관찰성 요구 사항에 맞는 보다 포괄적이고 유연하며 미래에 대비한 솔루션을 제공합니다. OpenTelemetry가 업계에서 광범위하게 채택하면 X-Ray와 직접 통합되지 않을 수 AWS 있는 외부 시스템 등 다양한 시스템에서 요청을 추적할 수 있습니다.

이 장에서는 원활한 전환을 위한 권장 사항을 제공하고 애플리케이션 계측 및 관찰성의 최신 기능에 대한 지속적인 지원과 액세스를 보장하기 위해 OpenTelemetry 기반 솔루션으로 마이그레이션하는 것의 중요성을 강조합니다.

OpenTelemetry를 애플리케이션 계측을 위한 관찰성 솔루션으로 채택하는 것이 좋습니다.

주제

OpenTelemetry 이해

OpenTelemetry는 원격 측정 데이터를 수집하기 위한 표준화된 프로토콜과 도구를 제공하는 업계 표준 관찰성 프레임워크입니다. 지표, 로그 및 추적과 같은 원격 측정 데이터를 계측, 생성, 수집 및 내보내는 통합 접근 방식을 제공합니다.

X-Ray SDKs OpenTelemetry 로 마이그레이션하면 다음과 같은 이점을 얻을 수 있습니다.

- 향상된 프레임워크 및 라이브러리 계측 지원

- 추가 프로그래밍 언어 지원
- 자동 계측 기능
- 유연한 샘플링 구성 옵션
- 지표, 로그 및 추적의 통합 수집

OpenTelemetry 수집기는 X-Ray 데몬보다 데이터 수집 형식 및 내보내기 대상에 대한 더 많은 옵션을 제공합니다.

에서 OpenTelemetry 지원 AWS

AWS 는 OpenTelemetry 작업을 위한 여러 솔루션을 제공합니다.

- AWS OpenTelemetry용 배포판

  OpenTelemetry 트레이스를 세그먼트로 X-Ray로 내보냅니다.

  자세한 내용은 AWS Distro for OpenTelemetry를 참조하세요.
- CloudWatch Application Signals

  사용자 지정 OpenTelemetry 트레이스 및 지표를 내보내 애플리케이션 상태를 모니터링합니다.

  자세한 내용은 Application Signals 작업을 참조하세요.
- CloudWatch OTel 엔드포인트

  기본 OpenTelemetry 계측과 함께 HTTP OTel 엔드포인트를 사용하여 OpenTelemetry 트레이스를 X-Ray로 내보냅니다.

  자세한 내용은 OTel 엔드포인트 사용을 참조하세요.

AWS CloudWatch에서 OpenTelemetry 사용

AWS CloudWatch는 클라이언트 측 애플리케이션 계측 및 Application Signals, Trace, Map, Metrics, Logs와 같은 네이티브 AWS CloudWatch 서비스를 통해 OpenTelemetry 추적을 지원합니다. 자세한 내용은 OpenTelemetry를 참조하세요.

마이그레이션을 위한 OpenTelemetry 개념 이해

다음 표에서는 X-Ray 개념을 OpenTelemetry에 매핑합니다. 이러한 매핑을 이해하면 기존 X-Ray 계측을 OpenTelemetry로 변환하는 데 도움이 됩니다.

X-Ray 개념	OpenTelemetry 개념
X-Ray 레코더	추적기 공급자 및 추적기
서비스 플러그인	리소스 감지기
세그먼트	(서버) 범위
하위 세그먼트	(서버 아님) 범위
X-Ray 샘플링 규칙	OpenTelemetry 샘플링(사용자 지정 가능)
X-Ray 이미터	Span Exporter(사용자 지정 가능)
주석/메타데이터	속성
라이브러리 계측	라이브러리 계측
X-Ray 트레이스 컨텍스트	범위 컨텍스트
X-Ray 트레이스 컨텍스트 전파	W3C 추적 컨텍스트 전파
X-Ray 트레이스 샘플링	OpenTelemetry 트레이스 샘플링
N/A	범위 처리
N/A	와이너리
X-Ray 데몬	OpenTelemetry Collector

> **Note**
>
> OpenTelemetry 개념에 대한 자세한 내용은 OpenTelemetry 설명서를 참조하세요.

기능 비교

다음 표에는 두 서비스에서 지원되는 기능이 나와 있습니다. 이 정보를 사용하여 마이그레이션 중에 해결해야 할 격차를 식별합니다.

Feature	X-Ray 계측	OpenTelemetry 계측
라이브러리 계측	지원	지원
X-Ray 샘플링	지원	OTel Java/에서 지원됩니다.NET/Go ADOT Java/에서 지원됩니다.NET/Python/Node.js
X-Ray 추적 컨텍스트 전파	지원	지원
리소스 감지	지원	지원
세그먼트 주석	지원	지원
세그먼트 메타데이터	지원	지원
제로 코드 자동 계측	Java에서 지원됨	OTel Java/에서 지원됩니다.NET/Python/Node.js ADOT Java/에서 지원됩니다.NET/Python/Node.js
수동으로 생성 추적	지원	지원

추적 설정 및 구성

OpenTelemetry에서 트레이스를 생성하려면 트레이서가 필요합니다. 애플리케이션에서 추적기 공급자를 초기화하여 추적기를 가져옵니다. 이는 X-Ray 레코더를 사용하여 X-Ray를 구성하고 X-Ray 트레이스에서 세그먼트 및 하위 세그먼트를 생성하는 방법과 유사합니다.

> **Note**
>
> OpenTelemetry Tracer Provider는 X-Ray 레코더보다 더 많은 구성 옵션을 제공합니다.

트레이스 데이터 구조 이해

기본 개념과 기능 매핑을 이해한 후 트레이스 데이터 구조 및 샘플링과 같은 특정 구현 세부 정보에 대해 알아볼 수 있습니다.

OpenTelemetry는 세그먼트 및 하위 세그먼트 대신 스팬을 사용하여 트레이스 데이터를 구성합니다. 각 스팬에는 다음 구성 요소가 포함됩니다.

- 명칭
- 고유 ID
- 시작 및 종료 타임스탬프
- 스팬 종류
- 범위 컨텍스트
- 속성(키-값 메타데이터)
- 이벤트(타임스탬프 처리된 로그)
- 다른 스팬에 대한 링크
- 상태 정보
- 상위 범위 참조

OpenTelemetry로 마이그레이션하면 스팬이 자동으로 X-Ray 세그먼트 또는 하위 세그먼트로 변환됩니다. 이렇게 하면 기존 CloudWatch 콘솔 환경이 변경되지 않습니다.

스팬 속성 작업

X-Ray SDK는 세그먼트 및 하위 세그먼트에 데이터를 추가하는 두 가지 방법을 제공합니다.

Annotations

필터링 및 검색을 위해 인덱싱되는 키-값 페어

메타데이터

검색을 위해 인덱싱되지 않은 복잡한 데이터가 포함된 키-값 페어

기본적으로 OpenTelemetry 스팬 속성은 X-Ray 원시 데이터의 메타데이터로 변환됩니다. 대신 특정 속성을 주석으로 변환하려면 해당 키를 `aws.xray.annotations` 속성 목록에 추가합니다.

- OpenTelemetry 개념에 대한 자세한 내용은 OpenTelemetry 추적을 참조하세요.
- OpenTelemetry 데이터가 X-Ray 데이터에 매핑되는 방법에 대한 자세한 내용은 OpenTelemetry에서 X-Ray 데이터 모델 변환을 참조하세요.

환경에서 리소스 감지

OpenTelemetry는 Resource Detectors를 사용하여 원격 측정 데이터를 생성하는 리소스에 대한 메타데이터를 수집합니다. 이 메타데이터는 리소스 속성으로 저장됩니다. 예를 들어 원격 측정을 생성하는 엔터티는 Amazon ECS 클러스터 또는 Amazon EC2 인스턴스일 수 있으며, 이러한 엔터티에서 기록할 수 있는 리소스 속성에는 Amazon ECS 클러스터 ARN 또는 Amazon EC2 인스턴스 ID가 포함될 수 있습니다.

- 지원되는 리소스 유형에 대한 자세한 내용은 OpenTelemetry 리소스 의미 체계 규칙을 참조하세요.
- X-Ray 서비스 플러그인에 대한 자세한 내용은 X-Ray SDK 구성을 참조하세요.

샘플링 전략 관리

트레이스 샘플링은 모든 요청 대신 대표적인 요청 하위 집합에서 데이터를 수집하여 비용을 관리하는 데 도움이 됩니다. OpenTelemetry와 X-Ray 모두 샘플링을 지원하지만 다르게 구현합니다.

> **Note**
>
> 트레이스를 100% 미만으로 샘플링하면 관찰성 비용이 절감되는 동시에 애플리케이션 성능에 대한 의미 있는 인사이트를 유지할 수 있습니다.

OpenTelemetry는 몇 가지 기본 제공 샘플링 전략을 제공하며 사용자 지정 샘플링 전략을 생성할 수 있습니다. OpenTelemetry에서 X-Ray 샘플링 규칙을 사용하도록 일부 SDK 언어로 X-Ray 원격 샘플러를 구성할 수도 있습니다.

OpenTelemetry의 추가 샘플링 전략은 다음과 같습니다.

- 상위 기반 샘플링 - 추가 샘플링 전략을 적용하기 전에 상위 범위의 샘플링 결정을 준수합니다.
- 트레이스 ID 비율 기반 샘플링 - >지정된 비율의 범위를 무작위로 샘플링합니다.

- 테일 샘플링 - 샘플링 규칙을 적용하여 OpenTelemetry Collector에서 트레이스를 완료합니다.
- 사용자 지정 샘플러 - 샘플링 인터페이스를 사용하여 자체 샘플링 로직 구현

X-Ray 샘플링 규칙에 대한 자세한 내용은 X-Ray 콘솔의 샘플링 규칙을 참조하세요.

OpenTelemetry 테일 샘플링에 대한 자세한 내용은 테일 샘플링 프로세서를 참조하세요.

추적 컨텍스트 관리

X-Ray SDKs 세그먼트 컨텍스트를 관리하여 트레이스에서 세그먼트와 하위 세그먼트 간의 상위-하위 관계를 올바르게 처리합니다. OpenTelemetry는 유사한 메커니즘을 사용하여 스팬에 올바른 상위 스팬이 있는지 확인합니다. 요청 컨텍스트 전체에 추적 데이터를 저장하고 전파합니다. 예를 들어 애플리케이션이 요청을 처리하고 해당 요청을 나타내는 서버 스팬을 생성하면 OpenTelemetry는 OpenTelemetry 컨텍스트에 서버 스팬을 저장하므로 하위 스팬이 생성될 때 해당 하위 스팬이 컨텍스트의 스팬을 상위로 참조할 수 있습니다.

추적 컨텍스트 전파

X-Ray 및 OpenTelemetry 모두 HTTP 헤더를 사용하여 서비스 간에 추적 컨텍스트를 전파합니다. 이를 통해 다양한 서비스에서 생성된 추적 데이터를 연결하고 샘플링 결정을 유지할 수 있습니다.

X-Ray SDK는 X-Ray 트레이스 헤더를 사용하여 트레이스 컨텍스트를 자동으로 전파합니다. 한 서비스가 다른 서비스를 호출하면 트레이스 헤더에는 트레이스 간 상위-하위 관계를 유지하는 데 필요한 컨텍스트가 포함됩니다.

OpenTelemetry는 컨텍스트 전파를 위해 다음을 포함한 여러 트레이스 헤더 형식을 지원합니다.

- W3C 추적 컨텍스트(기본값)
- X-Ray 트레이스 헤더
- 기타 사용자 지정 형식

> **Note**
>
> 하나 이상의 헤더 형식을 사용하도록 OpenTelemetry를 구성할 수 있습니다. 예를 들어 X-Ray 전파기를 사용하여 X-Ray 추적을 지원하는 AWS 서비스에 추적 컨텍스트를 전송합니다.

X-Ray 전파기를 구성하고 사용하여 AWS 서비스 간 추적을 활성화합니다. 이를 통해 트레이스 컨텍스트를 API Gateway 엔드포인트 및 X-Ray를 지원하는 기타 서비스에 전파할 수 있습니다.

- X-Ray 트레이스 헤더에 대한 자세한 내용은 X-Ray 개발자 안내서의 트레이스 헤더를 참조하세요.
- OpenTelemetry 컨텍스트 전파에 대한 자세한 내용은 OpenTelemetry 설명서의 컨텍스트 및 컨텍스트 전파를 참조하세요.

라이브러리 계측 사용

X-Ray와 OpenTelemetry 모두 애플리케이션에 추적을 추가하기 위해 최소한의 코드 변경이 필요한 라이브러리 계측을 제공합니다.

X-Ray는 라이브러리 계측 기능을 제공합니다. 이를 통해 애플리케이션 코드 변경을 최소화하면서 사전 구축된 X-Ray 계측을 추가할 수 있습니다. 이러한 계측은 AWS SDK 및 HTTP 클라이언트와 같은 특정 라이브러리와 Spring Boot 또는 Express.js와 같은 웹 프레임워크를 지원합니다.

OpenTelemetry의 계측 라이브러리는 라이브러리 후크 또는 자동 코드 수정을 통해 라이브러리에 대한 세부 범위를 생성하므로 코드 변경이 최소화됩니다.

OpenTelemetry의 Library Instrumentations가 라이브러리를 지원하는지 확인하려면 OpenTelemetry Registry의 OpenTelemetry Registry에서 라이브러리를 검색합니다.

트레이스 내보내기

X-Ray 및 OpenTelemetry는 다양한 방법을 사용하여 트레이스 데이터를 내보냅니다.

X-Ray 트레이스 내보내기

X-Ray SDKs 사용하여 추적 데이터를 전송합니다.

- 세그먼트 및 하위 세그먼트를 X-Ray 데몬으로 보냅니다.
- 비차단 I/O에 UDP 사용
- SDK에서 기본적으로 구성됨

OpenTelemetry 트레이스 내보내기

OpenTelemetry는 구성 가능한 Span Exporters를 사용하여 추적 데이터를 전송합니다.

- http/protobuf 또는 grpc 프로토콜 사용

- OpenTelemetry Collector 또는 CloudWatch Agent에서 모니터링하는 엔드포인트로 범위를 내보냅니다.
- 사용자 지정 내보내기 구성 허용

추적 처리 및 전달

X-Ray와 OpenTelemetry는 모두 추적 데이터를 수신, 처리 및 전달할 구성 요소를 제공합니다.

X-Ray 트레이스 처리

X-Ray 데몬은 트레이스 처리를 처리합니다.

- X-Ray SDKs에서 UDP 트래픽 수신 대기
- 세그먼트 및 하위 세그먼트 배치 처리
- X-Ray 서비스에 배치 업로드

OpenTelemetry 트레이스 처리

OpenTelemetry Collector는 추적 처리를 처리합니다.

- 계측된 서비스에서 트레이스를 수신합니다.
- 추적 데이터를 처리하고 선택적으로 수정합니다.
- 처리된 트레이스를 X-Ray를 포함한 다양한 백엔드로 전송합니다.

> **Note**
>
> AWS CloudWatch Agent는 OpenTelemetry 추적을 수신하여 X-Ray로 전송할 수도 있습니다. 자세한 내용은 OpenTelemetry를 사용하여 지표 및 추적 수집을 참조하세요.

스팬 처리(OpenTelemetry별 개념)

OpenTelemetry는 스팬 프로세서를 사용하여 스팬이 생성될 때 스팬을 수정합니다.

- 생성 또는 완료 시 스팬 읽기 및 수정 허용
- 범위 처리를 위한 사용자 지정 로직을 활성화합니다.

소행성(OpenTelemetry-soecific 개념)

OpenTelemetry의 도우미 기능을 사용하면 키-값 데이터를 전파할 수 있습니다.

- 추적 컨텍스트와 함께 임의의 데이터를 전달할 수 있습니다.
- 애플리케이션별 정보를 서비스 경계에 전파하는 데 유용합니다.

OpenTelemetry Collector에 대한 자세한 내용은 OpenTelemetry Collector를 참조하세요.

X-Ray 개념에 대한 자세한 내용은 X-Ray 개발자 안내서의 X-Ray 개념을 참조하세요.

마이그레이션 개요

이 섹션에서는 마이그레이션에 필요한 코드 변경 사항에 대한 개요를 제공합니다. 아래 목록은 언어별 지침 및 X-Ray 데몬 마이그레이션 단계입니다.

> **Important**
>
> X-Ray 계측에서 OpenTelemetry 계측으로 완전히 마이그레이션하려면 다음을 수행해야 합니다.
>
> 1. X-Ray SDK 사용량을 OpenTelemetry 솔루션으로 대체
> 2. X-Ray 데몬을 CloudWatch 에이전트 또는 OpenTelemetry Collector로 교체(X-Ray Exporter 사용)

- OpenTelemetry Java로 마이그레이션
- OpenTelemetry Go로 마이그레이션
- OpenTelemetry Node.js로 마이그레이션
- OpenTelemetry .NET으로 마이그레이션
- OpenTelemetry Python으로 마이그레이션
- OpenTelemetry Ruby로 마이그레이션

신규 및 기존 애플리케이션에 대한 권장 사항

신규 및 기존 애플리케이션의 경우 다음 솔루션을 사용하여 애플리케이션에서 추적을 활성화하는 것이 좋습니다.

계측

- OpenTelemetry SDKs
- AWS OpenTelemetry 계측용 배포판

데이터 수집

- OpenTelemetry Collector
- CloudWatch 에이전트

OpenTelemetry 기반 솔루션으로 마이그레이션한 후에도 CloudWatch 환경은 동일하게 유지됩니다. CloudWatch 콘솔의 트레이스 및 트레이스 맵 페이지에서 동일한 형식으로 트레이스를 보거나 X-Ray APIs를 통해 트레이스 데이터를 검색할 수 있습니다.

설정 변경 내용 추적

X-Ray 설정을 OpenTelemetry 설정으로 바꿔야 합니다.

X-Ray 및 OpenTelemetry 설정 비교

Feature	X-Ray SDK	OpenTelemetry
기본 구성	• X-Ray 중앙 집중식 샘플링 • X-Ray 추적 컨텍스트 전파 • X-Ray 데몬으로 트레이스 내보내기	• OpenTelemetry Collector 또는 CloudWatch Agent(HTTP/gRPC)로 트레이스 내보내기 • W3C 추적 컨텍스트 전파
수동 구성	• 로컬 샘플링 규칙 • 리소스 감지 플러그인	• X-Ray 샘플링(일부 언어에서는 제공되지 않을 수 있음) • 리소스 감지 • X-Ray 추적 컨텍스트 전파

라이브러리 계측 변경 사항

AWS SDK, HTTP 클라이언트, 웹 프레임워크 및 기타 라이브러리용 X-Ray 라이브러리 계측 대신 OpenTelemetry Library 계측을 사용하도록 코드를 업데이트합니다. 그러면 X-Ray 추적 대신 OpenTelemetry 추적이 생성됩니다.

> **Note**
>
> 코드 변경 사항은 언어와 라이브러리에 따라 다릅니다. 자세한 지침은 언어별 마이그레이션 가이드를 참조하세요.

Lambda 환경 계측 변경 사항

Lambda 함수에서 OpenTelemetry를 사용하려면 다음 설정 옵션 중 하나를 선택합니다.

1. 자동 계측 Lambda 계층 사용:
   - (권장) CloudWatch Application Signals Lambda 계층

     > **Note**
     >
     > 추적만 사용하려면 Lambda 환경 변수를 설정합니다 `OTEL_AWS_APPLICATION_SIGNALS_ENABLED=false`.

   - AWS ADOT용 관리형 Lambda 계층
2. Lambda 함수에 대한 OpenTelemetry를 수동으로 설정합니다.
   - X-Ray UDP 범위 내보내기를 사용하여 Simple Span 프로세서 구성
   - X-Ray Lambda 전파기 설정

추적 데이터 수동 생성

X-Ray 세그먼트 및 하위 세그먼트를 OpenTelemetry Spans로 바꿉니다.

- OpenTelemetry Tracer를 사용하여 스팬 생성
- Spans에 속성 추가(X-Ray 메타데이터 및 주석과 동일)

> **Important**
>
> X-Ray로 전송하는 경우:
>
> - 서버 범위를 X-Ray 세그먼트로 변환
> - 다른 스팬은 X-Ray 하위 세그먼트로 변환됩니다.

- 속성은 기본적으로 메타데이터로 변환됩니다.

속성을 주석으로 변환하려면 해당 키를 `aws.xray.annotations` 속성 목록에 추가합니다. 자세한 내용은 사용자 지정 X-Ray 주석 활성화를 참조하세요.

X-Ray 데몬에서 AWS CloudWatch 에이전트 또는 OpenTelemetry 수집기로 마이그레이션

CloudWatch 에이전트 또는 OpenTelemetry 수집기를 사용하여 계측된 애플리케이션에서 추적을 수신하여 X-Ray로 전송할 수 있습니다.

> **Note**
>
> CloudWatch 에이전트 버전 1.300025.0 이상은 OpenTelemetry 트레이스를 수집할 수 있습니다. X-Ray 데몬 대신 CloudWatch 에이전트를 사용하면 관리해야 하는 에이전트 수가 줄어듭니다. 자세한 내용은 CloudWatch 에이전트를 사용하여 지표, 로그 및 추적 수집을 참조하세요.

Sections

- Amazon EC2 또는 온프레미스 서버에서 마이그레이션
- Amazon ECS에서 마이그레이션
- Elastic Beanstalk에서 마이그레이션

Amazon EC2 또는 온프레미스 서버에서 마이그레이션

> **Important**
>
> 포트 충돌을 방지하기 위해 CloudWatch 에이전트 또는 OpenTelemetry 수집기를 사용하기 전에 X-Ray 데몬 프로세스를 중지합니다.

기존 X-Ray 데몬 설정

데몬 설치

기존 X-Ray 데몬 사용량은 다음 방법 중 하나를 사용하여 설치되었습니다.

수동 설치

X-Ray 데몬 Amazon S3 버킷에서 실행 파일을 다운로드하고 실행합니다.

자동 설치

이 스크립트를 사용하여 인스턴스를 시작할 때 데몬을 설치합니다.

```
#!/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
```

데몬 구성

기존 X-Ray 데몬 사용량은 다음 중 하나를 사용하여 구성되었습니다.

- 명령줄 인수
- 구성 파일(`xray-daemon.yaml`)

Example 구성 파일 사용

```
./xray -c ~/xray-daemon.yaml
```

데몬 실행

기존 X-Ray 데몬 사용이 다음 명령으로 시작되었습니다.

```
~/xray-daemon$ ./xray -o -n us-east-1
```

데몬 제거

Amazon EC2 인스턴스에서 X-Ray 데몬을 제거하려면:

1. 데몬 서비스를 중지합니다.

```
systemctl stop xray
```

2. 구성 파일을 삭제합니다.

```
rm ~/path/to/xray-daemon.yaml
```

3. 구성된 경우 로그 파일을 제거합니다.

> **Note**
>
> 로그 파일 위치는 구성에 따라 다릅니다.
>
> - 명령줄 구성: `/var/log/xray-daemon.log`
> - 구성 파일: `LogPath` 설정 확인

CloudWatch 에이전트 설정

에이전트 설치

설치 지침은 온프레미스 서버에 CloudWatch 에이전트 설치를 참조하세요.

에이전트 구성

1. 구성 파일을 생성하여 추적 수집을 활성화합니다. 자세한 내용은 CloudWatch 에이전트 구성 파일 생성을 참조하세요.
2. IAM 권한 설정:
   - IAM 역할을 연결하거나 에이전트의 자격 증명을 지정합니다. 자세한 내용은 IAM 역할 설정을 참조하세요.
   - 역할 또는 자격 증명에 `xray:PutTraceSegments` 권한이 포함되어 있는지 확인합니다.

에이전트 시작

에이전트를 시작하는 지침은 명령줄을 사용하여 CloudWatch 에이전트 시작을 참조하세요.

OpenTelemetry 수집기 설정

수집기 설치

운영 체제용 OpenTelemetry 수집기를 다운로드하여 설치합니다. 지침은 수집기 설치를 참조하세요.

수집기 구성

수집기에서 다음 구성 요소를 구성합니다.

- awsproxy 확장

  X-Ray 샘플링에 필요

- OTel 수신기

  애플리케이션에서 추적을 수집합니다.

- Xray 내보내기

  추적을 X-Ray로 전송합니다.

Example 샘플 수집기 구성 - 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**
>
> `xray:PutTraceSegments` 권한을 사용하여 AWS 자격 증명을 구성합니다. 자세한 내용은 자격 증명 지정을 참조하세요.

수집기 시작

구성 파일을 사용하여 수집기를 실행합니다.

```
otelcol --config=otel-collector-config.yaml
```

Amazon ECS에서 마이그레이션

> **Important**
>
> 작업 역할에는 사용하는 수집기에 대한 `xray:PutTraceSegments` 권한이 있어야 합니다. 포트 충돌을 방지하기 위해 동일한 호스트에서 CloudWatch 에이전트 또는 OpenTelemetry 수집기 컨테이너를 실행하기 전에 기존 X-Ray 데몬 컨테이너를 중지합니다.

CloudWatch 에이전트 사용

1. Amazon ECR 퍼블릭 갤러리에서 도커 이미지를 가져옵니다.
2. 라는 구성 파일을 생성합니다`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. Systems Manager Parameter Store에 구성을 저장합니다.

   1. https://console.aws.amazon.com/systems-manager/ 엽니다.
   2. 파라미터 생성을 선택합니다.
   3. 다음 값을 입력합니다.
      - 이름: `/ecs/cwagent/otel-config`
      - 티어: 표준
      - 유형: 문자열
      - 데이터 유형: 텍스트
      - 값: [여기에 cw-agent-otel.json 구성 붙여넣기]
4. 브리지 네트워크 모드를 사용하여 작업 정의를 생성합니다.

   사용하는 네트워크 모드에 따라 작업 정의의 구성이 달라집니다. 기본값은 브리지 네트워킹이며, 기본 VPC에서 사용할 수 있습니다. 브리지 네트워크에서 `OTEL_EXPORTER_OTLP_TRACES_ENDPOINT`환경 변수를 설정하여 OpenTelemetry SDK에 CloudWatch 에이전트의 엔드포인트와 포트를 알려줍니다. 또한 애플리케이션의 OpenTelemetry SDK에서 Collector 컨테이너로 트레이스를 전송하려면 애플리케이션 컨테이너에서 Collector 컨테이너로 연결되는 링크를 생성해야 합니다.

   Example CloudWatch 에이전트 작업 정의

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

자세한 내용은 Amazon [ECS에서 Amazon EC2 인스턴스 수준 지표를 수집하도록 CloudWatch 에이전트 배포를](#) 참조하세요.

OpenTelemetry 수집기 사용

1. Docker Hub`otel/opentelemetry-collector-contrib`에서 Docker 이미지를 가져옵니다. https://hub.docker.com/r/otel/opentelemetry-collector-contrib
2. Amazon EC2 수집기 구성 섹션에 표시된 것과 동일한 콘텐츠를 `otel-collector-config.yaml` 사용하여 라는 구성 파일을 생성하되 대신를 사용하도록 엔드포인트를 업데이트`0.0.0.0`합니다`127.0.0.1`.

3. Amazon ECS에서이 구성을 사용하려면 Systems Manager Parameter Store에 구성을 저장할 수 있습니다. 먼저 Systems Manager Parameter Store 콘솔로 이동하여 새 파라미터 생성을 선택합니다. 다음 정보를 사용하여 새 파라미터를 생성합니다.
   - 이름: /ecs/otel/config(이 이름은 수집기의 작업 정의에서 참조됨)
   - 티어: 표준
   - 유형: 문자열
   - 데이터 유형: 텍스트
   - 값: [여기에 otel-collector-config.yaml 구성 붙여넣기]
4. 브리지 네트워크 모드를 예로 사용하여 OpenTelemetry 수집기를 배포하는 작업 정의를 생성합니다.

   작업 정의에서 구성은 사용하는 네트워킹 모드에 따라 달라집니다. 기본값은 브리지 네트워킹이며, 기본 VPC에서 사용할 수 있습니다. 브리지 네트워크에서 OpenTelemetry SDK에 OpenTelemetry OpenTelemetry Collector의 엔드포인트와 포트를 알리도록 `OTEL_EXPORTER_OTLP_TRACES_ENDPOINT` 환경 변수를 설정합니다. 또한 애플리케이션의 OpenTelemetry SDK에서 Collector 컨테이너로 트레이스를 전송하려면 애플리케이션 컨테이너에서 Collector 컨테이너로 연결되는 링크를 생성해야 합니다.

   Example OpenTelemetry 수집기 작업 정의

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

Elastic Beanstalk에서 마이그레이션

> **Important**
>
> 포트 충돌을 방지하기 위해 CloudWatch 에이전트를 사용하기 전에 X-Ray 데몬 프로세스를 중지합니다.

기존 X-Ray 데몬 통합은 Elastic Beanstalk 콘솔을 사용하거나 구성 파일로 애플리케이션 소스 코드에서 X-Ray 데몬을 구성하여 활성화되었습니다.

CloudWatch 에이전트 사용

Amazon Linux 2 플랫폼에서 구성 파일을 사용하여 CloudWatch 에이전트를 `.ebextensions` 구성합니다.

1. 프로젝트 루트.ebextensions에 라는 디렉터리 생성
2. 디렉터리 cloudwatch.config 내에 다음 콘텐츠.ebextensions가 포함된 라는 파일을 생성합니다.

```
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. 배포 시 애플리케이션 소스 번들에 .ebextensions 디렉터리 포함

Elastic Beanstalk 구성 파일에 대한 자세한 내용은 구성 파일을 사용한 고급 환경 사용자 지정을 참조하세요.

OpenTelemetry Java로 마이그레이션

이 섹션에서는 X-Ray SDK에서 OpenTelemetry SDK for Java 애플리케이션으로 마이그레이션하는 방법에 대한 지침을 제공합니다.

Sections

- 제로 코드 자동 계측 솔루션
- SDK를 사용한 수동 계측 솔루션
- 수신 요청 추적(스프링 프레임워크 계측)

- AWS SDK v2 계측
- 발신 HTTP 호출 구성
- 다른 라이브러리에 대한 계측 지원
- 추적 데이터 수동 생성
- Lambda 계측

제로 코드 자동 계측 솔루션

With X-Ray Java agent

X-Ray Java 에이전트를 활성화하려면 애플리케이션의 JVM 인수를 수정해야 했습니다.

```
-javaagent:/path-to-disco/disco-java-agent.jar=pluginPath=/path-to-disco/disco-plugins
```

With OpenTelemetry-based Java agent

OpenTelemetry 기반 Java 에이전트를 사용합니다.

- ADOT Java 에이전트를 사용한 자동 계측에는 AWS Distro for OpenTelemetry(ADOT) Auto-Instrumentation Java 에이전트를 사용합니다. 자세한 내용은 Java 에이전트를 사용한 트레이스 및 지표에 대한 자동 계측을 참조하세요. 추적만 원하는 경우 `OTEL_METRICS_EXPORTER=none` 환경 변수를 비활성화합니다.는 Java 에이전트에서 지표를 내보냅니다.

  (선택 사항) ADOT Java 자동 계측 AWS 을 사용하여에서 애플리케이션을 자동으로 계측할 때 CloudWatch Application Signals를 활성화하여 현재 애플리케이션 상태를 모니터링하고 장기 애플리케이션 성능을 추적할 수도 있습니다. Application Signals는 애플리케이션, 서비스 및 종속성에 대한 통합된 애플리케이션 중심 보기를 제공하며 애플리케이션 상태를 모니터링하고 분류하는 데 도움이 됩니다. 자세한 내용은 Application Signals를 참조하세요.
- 자동 계측에 OpenTelemetry Java 에이전트를 사용합니다. 자세한 내용은 Java 에이전트를 사용한 제로 코드 계측을 참조하세요.

SDK를 사용한 수동 계측 솔루션

Tracing setup with X-Ray SDK

Java용 X-Ray SDK를 사용하여 코드를 계측하려면 먼저 AWSXRay 클래스를 서비스 플러그인 및 로컬 샘플링 규칙으로 구성한 다음 제공된 레코더를 사용해야 했습니다.

```
static { AWS XRayRecorderBuilder builder = AWS
 XRayRecorderBuilder.standard().withPlugin(new EC2Plugin()).withPlugin(new
 ECSPlugin()); AWS XRay.setGlobalRecorder(builder.build());
}
```

Tracing setup with OpenTelemetry SDK

다음 종속성이 필요합니다.

```
<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>
```

를 인스턴스화`TracerProvider`하고 `OpenTelemetrySdk` 객체를 전역적으로 등록하여 OpenTelemetry SDK를 구성합니다. 다음 구성 요소를 구성합니다.

- OTLP Span Exporter(예: OtlpGrpcSpanExporter) - CloudWatch 에이전트 또는 OpenTelemetry Collector로 트레이스를 내보내는 데 필요합니다.
- AWS X-Ray 전파기 - 추적 컨텍스트를 X-Ray와 통합된 AWS 서비스에 전파하는 데 필요합니다.
- AWS X-Ray 원격 샘플러 - X-Ray 샘플링 규칙을 사용하여 요청을 샘플링해야 하는 경우 필요합니다.
- 리소스 감지기(예: EcsResource 또는 Ec2Resource) - 애플리케이션을 실행하는 호스트의 메타데이터 감지

```
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();
```

수신 요청 추적(스프링 프레임워크 계측)

With X-Ray SDK

스프링 프레임워크와 함께 X-Ray SDK를 사용하여 애플리케이션을 계측하는 방법에 대한 자세한 내용은 Spring을 사용하는 AOP 및 Java용 X-Ray SDK를 참조하세요. Spring에서 AOP를 활성화하려면 다음 단계를 완료하세요.

1. Spring 구성
2. 애플리케이션에 추적 필터 추가
3. 코드에 주석 달기 또는 인터페이스 구현
4. 애플리케이션에서 X-Ray 활성화

With OpenTelemetry SDK

OpenTelemetry는 Spring Boot 애플리케이션에 대한 수신 요청에 대한 추적을 수집하는 계측 라이브러리를 제공합니다. 최소한의 구성으로 Spring Boot 계측을 활성화하려면 다음 종속성을 포함합니다.

```
<dependency>
           <groupId>io.opentelemetry.instrumentation</groupId>
            <artifactId>opentelemetry-spring-boot-starter</artifactId>
        </dependency>
```

OpenTelemetry 설정에 대해 Spring Boot 계측을 활성화하고 구성하는 방법에 대한 자세한 내용은 OpenTelemetry의 시작하기를 참조하세요.

Using OpenTelemetry-based Java agents

Spring Boot 애플리케이션을 계측하는 데 권장되는 기본 방법은 바이트코드 계측과 함께 OpenTelemetry Java 에이전트를 사용하는 것입니다.이 에이전트는 SDK를 직접 사용하는 경우와 비교할 때 더 많은 out-of-the-box 계측 및 구성을 제공합니다. 시작하기는 단원을 참조하십시오제로 코드 자동 계측 솔루션.

AWS SDK v2 계측

With X-Ray SDK

빌드에 `aws-xray-recorder-sdk-aws-sdk-v2-instrumentor` 하위 모듈을 추가하면 Java용 X-Ray SDK가 모든 AWS SDK v2 클라이언트를 자동으로 계측할 수 있습니다.

Java 2.2 이상용 AWS SDK를 사용하여 AWS 서비스에 대한 개별 클라이언트 다운스트림 클라이언트 호출을 계측하기 위해 빌드 구성의 `aws-xray-recorder-sdk-aws-sdk-v2-instrumentor` 모듈이 제외되고 `aws-xray-recorder-sdk-aws-sdk-v2` 모듈이 포함되었습니다. 개별 클라이언트는 로 구성하여 구성되었습니다`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

모든 AWS SDK 클라이언트를 자동으로 계측하려면 `opentelemetry-aws-sdk-2.2-autoconfigure` 하위 모듈을 추가합니다.

```
<dependency>
            <groupId>io.opentelemetry.instrumentation</groupId>
            <artifactId>opentelemetry-aws-sdk-2.2-autoconfigure</artifactId>
            <version>2.15.0-alpha</version>
            <scope>runtime</scope>
        </dependency>
```

개별 AWS SDK 클라이언트를 계측하려면 `opentelemetry-aws-sdk-2.2` 하위 모듈을 추가합니다.

```
<dependency>
            <groupId>io.opentelemetry.instrumentation</groupId>
            <artifactId>opentelemetry-aws-sdk-2.2</artifactId>
            <version>2.15.0-alpha</version>
            <scope>compile</scope>
        </dependency>
```

그런 다음 AWS SDK 클라이언트를 생성할 때 인터셉터를 등록합니다.

```
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();
```

발신 HTTP 호출 구성

With X-Ray SDK

X-Ray를 사용하여 발신 HTTP 요청을 계측하려면 Java용 X-Ray SDK의 Apache HttpClient 버전이 필요했습니다.

```
import com.amazonaws.xray.proxies.apache.http.HttpClientBuilder;
...
  public String randomName() throws IOException {
    CloseableHttpClient httpclient = HttpClientBuilder.create().build();
```

With OpenTelemetry SDK

X-Ray Java SDK와 마찬가지로 OpenTelemetry는 Apache HttpClient에 OpenTelemetry 기반 스팬 및 컨텍스트 전파를 제공하기 `HttpClientBuilder` 위해의 인스턴스를 생성할 수 있는 빌더 메서드가 있는 `ApacheHttpClientTelemetry` 클래스를 제공합니다.

```
<dependency>
            <groupId>io.opentelemetry.instrumentation</groupId>
            <artifactId>opentelemetry-apache-httpclient-5.2</artifactId>
            <version>2.15.0-alpha</version>
            <scope>compile</scope>
        </dependency>
```

다음은 opentelemetry-java-instrumentation 의 코드 예제입니다. newHttpClient()에서 제공하는 HTTP 클라이언트는 실행된 요청에 대한 추적을 생성합니다.

```
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();
  }
}
```

다른 라이브러리에 대한 계측 지원

지원되는 라이브러리, 프레임워크, 애플리케이션 서버 및 JVM 아래의 해당 계측 GitHub 리포지토리에서 OpenTelemetry Java에 지원되는 라이브러리 계측의 전체 목록을 찾습니다. JVMs

또는 OpenTelemetry 레지스트리를 검색하여 OpenTelemetry가 계측을 지원하는지 확인할 수 있습니다. 검색을 시작하려면 레지스트리를 참조하세요.

추적 데이터 수동 생성

With X-Ray SDK

X-Ray SDK를 사용하면 X-Ray 세그먼트 및 하위 세그먼트를 수동으로 생성하는 데 `beginSegment` 및 `beginSubsegment` 메서드가 필요합니다.

```
  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

사용자 지정 스팬을 사용하여 계측 라이브러리로 캡처되지 않은 내부 활동의 성능을 모니터링할 수 있습니다. 스팬 종류 서버만 X-Ray 세그먼트로 변환되고 다른 모든 스팬은 X-Ray 하위 세그먼트로 변환됩니다.

먼저`openTelemetry.getTracer`, 메서드를 통해 얻을 수 있는 스팬을 생성하려면 트레이서를 생성해야 합니다. 그러면 SDK를 사용한 수동 계측 솔루션 예제에 전역적으로 `TracerProvider` 등록된의 Tracer 인스턴스가 제공됩니다. 필요한 만큼 Tracer 인스턴스를 생성할 수 있지만 전체 애플리케이션에 대해 하나의 Tracer를 사용하는 것이 일반적입니다.

```
Tracer tracer = openTelemetry.getTracer("my-app");
```

추적기를 사용하여 스팬을 생성할 수 있습니다.

```
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();
```

스팬의 기본 유형은 내부입니다.

```
// Default span of type INTERNAL will become an X-Ray subsegment
Span span = tracer.spanBuilder("process-header")
  .startSpan();
try (Scope ignored = span.makeCurrent()) {
  doProcessHeader();
}
```

OpenTelemetry SDK를 사용하여 트레이스에 주석 및 메타데이터 추가

위 예제에서 `setAttribute` 메서드는 각 스팬에 속성을 추가하는 데 사용됩니다. 기본적으로 모든 스팬 속성은 X-Ray 원시 데이터의 메타데이터로 변환됩니다. 속성이 메타데이터가 아닌 주석으로 변환되도록 하기 위해 위 예제에서는 해당 속성의 키를 `aws.xray.annotations` 속성 목록에 추가합니다. 자세한 내용은 사용자 지정 X-Ray 주석 및 주석 및 메타데이터 활성화를 참조하세요.

OpenTelemetry 기반 Java 에이전트 사용

Java 에이전트를 사용하여 애플리케이션을 자동으로 계측하는 경우 애플리케이션에서 수동 계측을 수행해야 합니다. 예를 들어 자동 계측 라이브러리에서 다루지 않는 섹션의 애플리케이션 내에서 코드를 계측하는 것입니다.

에이전트를 사용하여 수동 계측을 수행하려면 `opentelemetry-api` 아티팩트를 사용해야 합니다. 아티팩트 버전은 에이전트 버전보다 최신일 수 없습니다.

```
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 계측

With X-Ray SDK

X-Ray SDK를 사용하면 Lambda가 활성 추적을 활성화한 후 X-Ray SDK를 사용하는 데 추가 구성이 필요하지 않습니다. Lambda는 Lambda 핸들러 호출을 나타내는 세그먼트를 생성하며, 추가 구성 없이 X-Ray SDK를 사용하여 하위 세그먼트 또는 계측 라이브러리를 생성할 수 있습니다.

With OpenTelemetry-based solutions

자동 계측 Lambda 계층 - 다음 솔루션을 사용하여 AWS 벤딩된 Lambda 계층으로 Lambda를 자동으로 계측할 수 있습니다.

- CloudWatch Application Signals Lambda 계층(권장)

> **Note**
>
> 이 Lambda 계층에는 기본적으로 CloudWatch Application Signals가 활성화되어 있으므로 지표와 트레이스를 모두 수집하여 Lambda 애플리케이션에 대한 성능 및 상태 모니터링을 사용할 수 있습니다. 추적만 하려면 Lambda 환경 변수를 설정합니다 `OTEL_AWS_APPLICATION_SIGNALS_ENABLED=false`.

  - Lambda 애플리케이션에 대한 성능 및 상태 모니터링을 활성화합니다.
  - 기본적으로 지표와 추적을 모두 수집합니다.
- AWS ADOT Java용 관리형 Lambda 계층입니다. 자세한 내용은 AWS Java에 대한 Distro for OpenTelemetry Lambda Support를 참조하세요.

자동 계측 계층과 함께 수동 계측을 사용하려면 섹션을 참조하세요SDK를 사용한 수동 계측 솔루션. 콜드 스타트를 줄이려면 OpenTelemetry 수동 계측을 사용하여 Lambda 함수에 대한 OpenTelemetry 트레이스를 생성하는 것이 좋습니다.

AWS Lambda용 OpenTelemetry 수동 계측

Amazon S3 ListBuckets를 호출하는 다음 Lambda 함수 코드를 고려합니다.

```
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);
        }
    }
}
```

종속성은 다음과 같습니다.

```
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')
}
```

Lambda 핸들러와 Amazon S3 클라이언트를 수동으로 계측하려면 다음을 수행합니다.

1. `RequestHandler` (또는 RequestStreamHandler)를 구현하는 함수 클래스를 `TracingRequestHandler` (또는 TracingRequestStreamHandler)를 확장하는 함수 클래스로 바꿉니다.
2. TracerProvider를 인스턴스화하고 OpenTelemetrySdk 객체를 전역적으로 등록합니다. TracerProvider는 다음과 같이 구성하는 것이 좋습니다.
   a. 추적을 Lambda의 UDP X-Ray 엔드포인트로 전송하기 위한 X-Ray UDP 스팬 내보내기가 있는 Simple Span 프로세서
   b. ParentBased 상시 기반 샘플러(구성되지 않은 경우 기본값)
   c. service.name Lambda 함수 이름으로 설정된 리소스
   d. X-Ray Lambda 전파기
3. `handleRequest` 메서드를 로 변경`doHandleRequest`하고 `OpenTelemetrySdk` 객체를 기본 클래스에 전달합니다.
4. 클라이언트를 빌드할 때 인터셉터를 등록하여 OpenTemetry AWS SDK 계측으로 Amazon S3 클라이언트를 계측합니다.

다음과 같은 OpenTelemetry 관련 종속성이 필요합니다.

```
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')
}
```

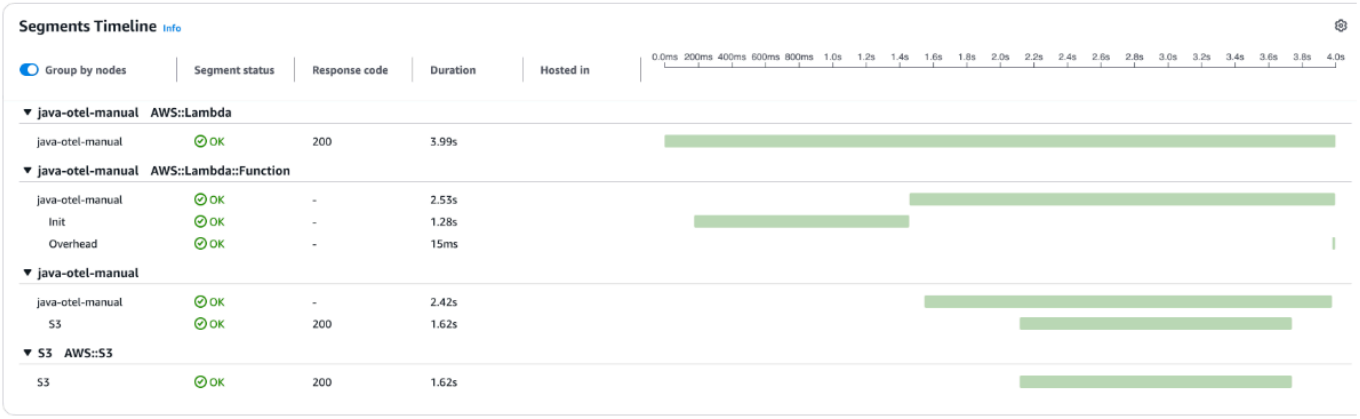
다음 코드는 필요한 변경 후 Lambda 함수를 보여줍니다. 추가 사용자 지정 스팬을 생성하여 자동으로 제공되는 스팬을 보완할 수 있습니다.

```
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);
        }
    }
}
```

Lambda 함수를 호출할 때 CloudWatch 콘솔의 트레이스 맵 아래에 다음 트레이스가 표시됩니다.



OpenTelemetry Go로 마이그레이션

다음 코드 예제를 사용하여 X-Ray에서 마이그레이션할 때 OpenTelemetry SDK를 사용하여 Go 애플리케이션을 수동으로 계측할 수 있습니다.

SDK를 사용한 수동 계측

Tracing setup with X-Ray SDK

Go용 X-Ray SDK를 사용하는 경우 코드를 계측하기 전에 서비스 플러그인 또는 로컬 샘플링 규칙을 구성해야 했습니다.

```
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

TracerProvider를 인스턴스화하고 글로벌 추적기 공급자로 등록하여 OpenTelemetry SDK를 구성합니다. 다음 구성 요소를 구성하는 것이 좋습니다.

- OTLP 추적 내보내기 - CloudWatch 에이전트 또는 OpenTelemetry Collector로 추적을 내보내는 데 필요합니다.
- X-Ray 전파기 - 추적 컨텍스트를 X-Ray와 통합된 AWS 서비스로 전파하는 데 필요합니다.
- X-Ray 원격 샘플러 - X-Ray 샘플링 규칙을 사용하여 요청을 샘플링하는 데 필요합니다.
- 리소스 감지기 - 애플리케이션을 실행하는 호스트의 메타데이터 감지

```
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
}
```

수신 요청 추적(HTTP 핸들러 계측)

With X-Ray SDK

X-Ray로 HTTP 핸들러를 계측하기 위해 X-Ray 핸들러 메서드를 사용하여 NewFixedSegmentNamer를 사용하여 세그먼트를 생성했습니다.

```
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

OpenTelemetry로 HTTP 핸들러를 계측하려면 OpenTelemetry의 newHandler 메서드를 사용하여 원래 핸들러 코드를 래핑합니다.

```
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 SDK for Go v2 계측

With X-Ray SDK

AWS SDK의 발신 AWS 요청을 계측하기 위해 클라이언트는 다음과 같이 계측되었습니다.

```
// 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

다운스트림 AWS SDK 호출에 대한 추적 지원은 OpenTelemetry의 AWS SDK for Go v2 Instrumentation에서 제공합니다. 다음은 S3 클라이언트 호출을 추적하는 예입니다.

```
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
    }
```

발신 HTTP 호출 구성

With X-Ray SDK

X-Ray를 사용하여 발신 HTTP 호출을 계측하기 위해 xray.Client를 사용하여 제공된 HTTP 클라이언트의 복사본을 생성했습니다.

```
myClient := xray.Client(http-client)

resp, err := ctxhttp.Get(ctx, xray.Client(nil), url)
```

With OpenTelemetry SDK

OpenTelemetry를 사용하여 HTTP 클라이언트를 계측하려면 OpenTelemetry의 otelhttp.NewTransport 메서드를 사용하여 http.DefaultTransport를 래핑합니다.

```
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)
```

다른 라이브러리에 대한 계측 지원

OpenTelemetry Go에 지원되는 라이브러리 계측의 전체 목록은 계측 패키지에서 확인할 수 있습니다.

또는 OpenTelemetry 레지스트리를 검색하여 OpenTelemetry가 레지스트리에서 라이브러리에 대한 계측을 지원하는지 확인할 수 있습니다.

추적 데이터 수동 생성

With X-Ray SDK

X-Ray SDK를 사용하면 X-Ray 세그먼트 및 하위 세그먼트를 수동으로 생성하려면 BeginSegment 및 BeginSubsegment 메서드가 필요했습니다.

```
// 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

사용자 지정 스팬을 사용하여 계측 라이브러리로 캡처되지 않은 내부 활동의 성능을 모니터링합니다. 참고로 서버는 X-Ray 세그먼트로 변환되고 다른 모든 스팬은 X-Ray 하위 세그먼트로 변환됩니다.

먼저 추적기를 생성하여 `otel.Tracer` 메서드를 통해 얻을 수 있는 스팬을 생성해야 합니다. 그러면 추적 설정 예제에서 전역적으로 등록된 TracerProvider 인스턴스가 제공됩니다. 필요한 만큼 Tracer 인스턴스를 생성할 수 있지만 전체 애플리케이션에 대해 하나의 Tracer를 사용하는 것이 일반적입니다.

```
    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()
```

OpenTelemetry SDK를 사용하여 트레이스에 주석 및 메타데이터 추가

위 예제에서 `WithAttributes` 메서드는 각 스팬에 속성을 추가하는 데 사용됩니다. 기본적으로 모든 스팬 속성은 X-Ray 원시 데이터의 메타데이터로 변환됩니다. 속성이 메타데이터가 아닌 주석으로 변환되도록 하려면 속성 목록에 `aws.xray.annotations` 속성의 키를 추가합니다. 자세한 내용은 사용자 지정 X-Ray 주석 활성화를 참조하세요.

Lambda 수동 계측

With X-Ray SDK

X-Ray SDK를 사용하면 Lambda에 활성 추적이 활성화된 후 X-Ray SDK를 사용하는 데 추가 구성이 필요하지 않았습니다. Lambda는 Lambda 핸들러 호출을 나타내는 세그먼트를 생성했으며 추가 구성 없이 X-Ray SDK를 사용하여 하위 세그먼트를 생성했습니다.

With OpenTelemetry SDK

다음 Lambda 함수 코드(계측 없음)는 Amazon S3 ListBuckets 호출 및 발신 HTTP 요청을 수행합니다.

```
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)
}
```

Lambda 핸들러와 Amazon S3 클라이언트를 수동으로 계측하려면 다음을 수행합니다.

1. main()에서 TracerProvider(tp)를 인스턴스화하고 글로벌 추적기 공급자로 등록합니다. TracerProvider는 다음과 같이 구성하는 것이 좋습니다.
   a. 추적을 Lambda의 UDP X-Ray 엔드포인트로 전송하기 위한 X-Ray UDP 스팬 내보내기가 있는 Simple Span Processor
   b. service.name Lambda 함수 이름으로 설정된 리소스
2. 의 사용량을 `lambda.Start(lambdaHandler)`로 변경합니다`lambda.Start(otellambda.InstrumentHandler(lambdaHandler, xrayconfig.WithRecommendedOptions(tp)...))`.
3. 용 OpenTemetry AWS 미들웨어를 Amazon S3 클라이언트 구성에 추가하여 OpenTelemetry SDK 계측`aws-sdk-go-v2`으로 Amazon S3 클라이언트를 계측합니다.
4. OpenTelemetry의 `otelhttp.NewTransport` 메서드를 사용하여를 래핑하여 http 클라이언트를 계측합니다`http.DefaultTransport`.

다음 코드는 변경 후 Lambda 함수가 어떻게 표시되는지 보여주는 예입니다. 자동으로 제공되는 스팬 외에도 추가 사용자 지정 스팬을 수동으로 생성할 수 있습니다.

```
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)...))
```

```
}
```

Lambda를 호출할 때 CloudWatch 콘솔의 `Trace Map`에 다음 추적이 표시됩니다.

OpenTelemetry Node.js로 마이그레이션

이 섹션에서는 Node.js 애플리케이션을 X-Ray SDK에서 OpenTelemetry로 마이그레이션하는 방법을 설명합니다. 자동 및 수동 계측 접근 방식을 모두 다루며 일반적인 사용 사례에 대한 구체적인 예를 제공합니다.

X-Ray Node.js SDK를 사용하면 추적을 위해 Node.js 애플리케이션을 수동으로 계측할 수 있습니다. 이 섹션에서는 X-Ray에서 OpenTelemetry 계측으로 마이그레이션하기 위한 코드 예제를 제공합니다.

Sections

- 제로 코드 자동 계측 솔루션
- 수동 계측 솔루션
- 수신 요청 추적
- AWS SDK JavaScript V3 계측
- 발신 HTTP 호출 구성
- 다른 라이브러리에 대한 계측 지원
- 추적 데이터 수동 생성
- Lambda 계측

제로 코드 자동 계측 솔루션

Node.js용 X-Ray SDK를 사용하여 요청을 추적하려면 애플리케이션 코드를 수정해야 합니다. OpenTelemetry를 사용하면 제로 코드 자동 계측 솔루션을 사용하여 요청을 추적할 수 있습니다.

OpenTelemetry 기반 자동 계측을 사용한 제로 코드 자동 계측.

1. Node.js용 AWS Distro for OpenTelemetry(ADOT) 자동 계측 사용 - Node.js 애플리케이션에 대한 자동 계측은 AWS Distro for OpenTelemetry JavaScript 자동 계측을 사용한 추적 및 지표를 참조하세요.

   (선택 사항) ADOT JavaScript 자동 계측 AWS 을 사용하여에서 애플리케이션을 자동으로 계측할 때 CloudWatch Application Signals를 활성화하여 현재 애플리케이션 상태를 모니터링하고 비즈니스 목표에 따라 장기 애플리케이션 성능을 추적할 수도 있습니다. Application Signals는 애플리케이션, 서비스 및 종속성에 대한 통합 애플리케이션 중심 보기를 제공하고 애플리케이션 상태를 모니터링하고 분류하는 데 도움이 됩니다. 자세한 내용은 Application Signals를 참조하세요.
2. OpenTelemetry JavaScript 제로 코드 자동 계측 사용 - OpenTelemetry JavaScript를 사용한 자동 계측은 JavaScript 제로 코드 계측을 참조하세요.

수동 계측 솔루션

Tracing setup with X-Ray SDK

Node.js용 X-Ray SDK를 사용할 때 SDK를 사용하여 코드를 계측하기 전에 `aws-xray-sdk` 패키지가 서비스 플러그인 또는 로컬 샘플링 규칙으로 X-Ray SDK를 구성해야 했습니다.

```
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 원격 샘플링은 현재 OpenTelemetry JS에 대해 구성할 수 없습니다. 그러나 X-Ray 원격 샘플링에 대한 지원은 현재 Node.js용 ADOT 자동 계측을 통해 제공됩니다.

아래 코드 예제의 경우 다음 종속성이 필요합니다.

```
npm install --save \
  @opentelemetry/api \
  @opentelemetry/sdk-node \
  @opentelemetry/exporter-trace-otlp-proto \
  @opentelemetry/propagator-aws-xray \
  @opentelemetry/resource-detector-aws
```

애플리케이션 코드를 실행하기 전에 OpenTelemetry SDK를 설정하고 구성해야 합니다. 이 작업은 --require 플래그를 사용하여 수행할 수 있습니다. OpenTelemetry 계측 구성 및 설정이 포함된 instrumentation.js라는 파일을 생성합니다.

다음 구성 요소를 구성하는 것이 좋습니다.

- OTLPTraceExporter - CloudWatch 에이전트/OpenTelemetry Collector로 트레이스를 내보내는 데 필요합니다.
- AWSXRayPropagator - 추적 컨텍스트를 X-Ray와 통합된 AWS 서비스에 전파하는 데 필요합니다.
- Resource Detectors(예: Amazon EC2 Resource Detector) - 애플리케이션을 실행하는 호스트의 메타데이터를 감지합니다.

```
/*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();
```

그런 다음 다음과 같은 OpenTelemetry 설정을 사용하여 애플리케이션을 실행할 수 있습니다.

```
node --require ./instrumentation.js app.js
```

OpenTelemetry SDK 라이브러리 계측을 사용하여 AWS SDK와 같은 라이브러리에 대한 스팬을 자동으로 생성할 수 있습니다. 이를 활성화하면 AWS SDK for JavaScript v3와 같은 모듈에 대한 스팬이 자동으로 생성됩니다. OpenTelemetry는 모든 라이브러리 계측을 활성화하거나 활성화할 라이브러리 계측을 지정하는 옵션을 제공합니다.

모든 계측을 활성화하려면 `@opentelemetry/auto-instrumentations-node` 패키지를 설치합니다.

```
npm install @opentelemetry/auto-instrumentations-node
```

그런 다음 아래와 같이 구성을 업데이트하여 모든 라이브러리 계측을 활성화합니다.

```
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

Node.js용 ADOT 자동 계측을 사용하여 Node.js 애플리케이션에 대해 OpenTelemetry를 자동으로 구성할 수 있습니다. ADOT Auto-Instrumentation을 사용하면 수신 요청을 추적하거나 AWS SDK 또는 HTTP 클라이언트와 같은 라이브러리를 추적하기 위해 수동으로 코드를 변경할 필요가 없습니다. 자세한 내용은 AWS Distro for OpenTelemetry JavaScript Auto-Instrumentation을 사용한 추적 및 지표를 참조하세요.

Node.js용 ADOT 자동 계측은 다음을 지원합니다.

- 환경 변수를 통한 X-Ray 원격 샘플링 - `export OTEL_TRACES_SAMPLER=xray`
- X-Ray 추적 컨텍스트 전파(기본적으로 활성화됨)
- 리소스 감지(Amazon EC2, Amazon ECS 및 Amazon EKS 환경에 대한 리소스 감지는 기본적으로 활성화되어 있음)
- `OTEL_NODE_ENABLED_INSTRUMENTATIONS` 및 `OTEL_NODE_DISABLED_INSTRUMENTATIONS` 환경 변수를 통해 선택적으로 비활성화/활성화할 수 있는 지원되는 모든 OpenTelemetry 계측에 대한 자동 라이브러리 계측
- 스팬 수동 생성

수신 요청 추적

With X-Ray SDK

Express.js

Express.js 애플리케이션에서 수신한 수신 HTTP 요청을 추적하기 위한 X-Ray SDK를 사용하면 두 미들웨어 `AWSXRay.express.openSegment(<name>)` 및 `AWSXRay.express.closeSegment()`가 이를 추적하기 위해 정의된 모든 경로를 래핑해야 했습니다.

```
app.use(xrayExpress.openSegment('defaultName'));

...

app.use(xrayExpress.closeSegment());
```

재지정

`Restify` 애플리케이션에서 수신한 HTTP 요청을 추적하기 위해 Restify 서버의 `aws-xray-sdk-restify` 모듈에서 활성화를 실행하여 X-Ray SDK의 미들웨어를 사용했습니다.

```
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

에 대한 수신 요청에 대한 추적 지원`Express.js`은 OpenTelemetry HTTP 계측 및 OpenTelemetry Express 계측을 통해 제공됩니다. 를 사용하여 `npm`다음 종속성을 설치합니다.

```
npm install --save @opentelemetry/instrumentation-http @opentelemetry/
instrumentation-express
```

Express 모듈에 대한 계측을 활성화하도록 OpenTelemetry SDK 구성을 업데이트합니다.

```
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(),
  ],
});
```

재지정

Restify 애플리케이션의 경우 OpenTelemetry Restify 계측이 필요합니다. 다음 종속성을 설치합니다.

```
npm install --save @opentelemetry/instrumentation-restify
```

OpenTelemetry SDK 구성을 업데이트하여 재지정 모듈에 대한 계측을 활성화합니다.

```
const { RestifyInstrumentation } = require('@opentelemetry/instrumentation-
restify');
...

const sdk = new NodeSDK({
  ...

  instrumentations: [
    ...
    new RestifyInstrumentation(),
  ],
});
```

AWS SDK JavaScript V3 계측

With X-Ray SDK

AWS SDK의 발신 AWS 요청을 계측하기 위해 다음 예제와 같이 클라이언트를 계측했습니다.

```
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

DynamoDB, Amazon S3 등에 대한 다운스트림 AWS SDK 호출에 대한 추적 지원은 OpenTelemetry AWS SDK 계측을 통해 제공됩니다. 를 사용하여 npm다음 종속성을 설치합니다.

```
npm install --save @opentelemetry/instrumentation-aws-sdk
```

SDK 계측을 사용하여 OpenTelemetry AWS SDK 구성을 업데이트합니다.

```
import { AwsInstrumentation } from '@opentelemetry/instrumentation-aws-sdk';
...

const sdk = new NodeSDK({
  ...

  instrumentations: [
    ...
    new AwsInstrumentation()
  ],
});
```

발신 HTTP 호출 구성

With X-Ray SDK

X-Ray를 사용하여 발신 HTTP 요청을 계측하려면 클라이언트를 계측해야 했습니다. 예를 들어 아래를 참조하세요.

개별 HTTP 클라이언트

```
var AWSXRay = require('aws-xray-sdk');
var http = AWSXRay.captureHTTPs(require('http'));
```

모든 HTTP 클라이언트(글로벌)

```
var AWSXRay = require('aws-xray-sdk');
AWSXRay.captureHTTPsGlobal(require('http'));
var http = require('http');
```

With OpenTelemetry SDK

Node.js HTTP 클라이언트에 대한 추적 지원은 OpenTelemetry HTTP Instrumentation에서 제공합니다. 를 사용하여 npm다음 종속성을 설치합니다.

```
npm install --save @opentelemetry/instrumentation-http
```

다음과 같이 OpenTelemetry SDK 구성을 업데이트합니다.

```
const { HttpInstrumentation } = require('@opentelemetry/instrumentation-http');
...

const sdk = new NodeSDK({
  ...

  instrumentations: [
    ...
    new HttpInstrumentation(),
  ],
});
```

다른 라이브러리에 대한 계측 지원

OpenTelemetry JavaScript에 대해 지원되는 라이브러리 계측의 전체 목록은 지원되는 계측에서 확인할 수 있습니다.

또는 OpenTelemetry 레지스트리를 검색하여 OpenTelemetry가 레지스트리에서 라이브러리에 대한 계측을 지원하는지 확인할 수 있습니다.

추적 데이터 수동 생성

With X-Ray SDK

X-Ray를 사용하면 세그먼트와 하위 하위 세그먼트를 수동으로 생성하여 애플리케이션을 추적하는 데 `aws-xray-sdk` 패키지 코드가 필요했습니다.

```
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

사용자 지정 스팬을 생성하고 사용하여 계측 라이브러리로 캡처되지 않은 내부 활동의 성능을 모니터링할 수 있습니다. 참고로 서버는 X-Ray 세그먼트로 변환되고 다른 모든 스팬은 X-Ray 하위 세그먼트로 변환됩니다. 자세한 내용은 세그먼트를 참조하세요.

추적 설정에서 OpenTelemetry SDK를 구성하여 스팬을 생성한 후에는 추적기 인스턴스가 필요합니다. 필요한 만큼 Tracer 인스턴스를 생성할 수 있지만 전체 애플리케이션에 대해 하나의 Tracer를 사용하는 것이 일반적입니다.

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

OpenTelemetry SDK를 사용하여 트레이스에 주석 및 메타데이터 추가

사용자 지정 키-값 페어를 스팬의 속성으로 추가할 수도 있습니다. 기본적으로 이러한 모든 스팬 속성은 X-Ray 원시 데이터의 메타데이터로 변환됩니다. 속성이 메타데이터가 아닌 주석으로 변환되도록 하려면 속성 목록에 `aws.xray.annotations` 속성의 키를 추가합니다. 자세한 내용은 사용자 지정 X-Ray 주석 활성화를 참조하세요.

```
  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 계측

With X-Ray SDK

Lambda 함수에 대해 활성 추적을 활성화한 후 추가 구성 없이 X-Ray SDK가 필요했습니다. Lambda는 Lambda 핸들러 호출을 나타내는 세그먼트를 생성하고 추가 구성 없이 X-Ray SDK를 사용하여 하위 세그먼트 또는 계측 라이브러리를 생성했습니다.

With OpenTelemetry SDK

AWS 벤딩된 Lambda 계층으로 Lambda를 자동으로 계측할 수 있습니다. 두 가지 해결 방법이 있습니다.

- (권장) CloudWatch Application Signals lambda 계층

> **Note**
>
> 이 Lambda 계층에는 CloudWatch Application Signals가 기본적으로 활성화되어 있으므로 지표와 트레이스를 모두 수집하여 Lambda 애플리케이션에 대한 성능 및 상태 모니터링을 사용할 수 있습니다. 추적만 원하는 경우 Lambda 환경 변수를 설정해야 합니다`OTEL_AWS_APPLICATION_SIGNALS_ENABLED=false`. 자세한 내용은 Lambda에서 애플리케이션 활성화를 참조하세요.

- AWS ADOT JS용 관리형 Lambda 계층입니다. 자세한 내용은 AWS JavaScript에 대한 OpenTelemetry Lambda 지원을 위한 배포를 참조하세요.

Lambda 계측을 사용하여 수동으로 스팬 생성

ADOT JavaScript Lambda 계층은 Lambda 함수에 대한 자동 계측을 제공하지만, 예를 들어 사용자 지정 데이터를 제공하거나 라이브러리 계측에서 다루지 않는 Lambda 함수 자체 내의 코드를 계측하기 위해 Lambda에서 수동 계측을 수행해야 할 수 있습니다.

자동 계측과 함께 수동 계측을 수행하려면를 종속성`@opentelemetry/api`으로 추가해야 합니다. 이 종속성의 버전은 ADOT JavaScript SDK에서 사용하는 것과 동일한 종속성의 버전인 것이 좋습니다. OpenTelemetry API를 사용하여 Lambda 함수에서 스팬을 수동으로 생성할 수 있습니다.

NPM을 사용하여 `@opentelemetry/api` 종속성을 추가하려면:

```
npm install @opentelemetry/api
```

OpenTelemetry .NET으로 마이그레이션

.NET 애플리케이션에서 X-Ray 추적을 사용하는 경우 수동 작업이 포함된 X-Ray .NET SDK가 계측에 사용됩니다.

이 섹션에서는 X-Ray 수동 계측 솔루션에서 .NET용 OpenTelemetry 수동 계측 솔루션으로 마이그레이션하기 위한 SDK를 사용한 수동 계측 솔루션 섹션의 코드 예제를 제공합니다. 또는 제로 코드 자동 계측 솔루션 섹션에서 애플리케이션 소스 코드를 수정할 필요 없이 X-Ray 수동 계측에서 OpenTelemetry 자동 계측 솔루션으로 마이그레이션하여 .NET 애플리케이션을 계측할 수 있습니다.

Sections

제로 코드 자동 계측 솔루션

OpenTelemetry는 제로 코드 자동 계측 솔루션을 제공합니다. 이러한 솔루션은 애플리케이션 코드를 변경할 필요 없이 요청을 추적합니다.

OpenTelemetry 기반 자동 계측 옵션

1. .NET용 AWS Distro for OpenTelemetry(ADOT) 자동 계측 사용 - .NET 애플리케이션을 자동으로 계측하려면 AWS Distro for OpenTelemetry .NET Auto-Instrumentation을 사용하여 추적 및 지표를 참조하세요.

   (선택 사항) ADOT .NET 자동 계측 AWS 을 사용하여에서 애플리케이션을 자동으로 계측할 때 CloudWatch Application Signals를 활성화하여 다음을 수행합니다.

   - 현재 애플리케이션 상태 모니터링

- 비즈니스 목표를 기준으로 장기 애플리케이션 성능 추적
- 애플리케이션, 서비스 및 종속성에 대한 통합된 애플리케이션 중심 보기 가져오기
- 애플리케이션 상태 모니터링 및 분류

자세한 내용은 Application Signals를 참조하세요.

2. OpenTelemetry .Net 제로 코드 자동 계측 사용 - OpenTelemetry .Net을 사용하여 자동으로 계측하려면 AWS Distro for OpenTelemetry .NET Auto-Instrumentation을 사용하여 추적 및 지표를 참조하세요.

SDK를 사용한 수동 계측 솔루션

Tracing configuration with X-Ray SDK

.NET 웹 애플리케이션의 경우 X-Ray SDK는 `Web.config` 파일의 appSettings 섹션에 구성됩니다.

Web.config 예제

```
<configuration>
  <appSettings>
    <add key="AWSXRayPlugins" value="EC2Plugin"/>
  </appSettings>
</configuration>
```

.NET Core의 경우 라는 최상위 키가 `appsettings.json` 있는 라는 파일이 `XRay` 사용된 다음 X-Ray 레코더를 초기화하기 위해 구성 객체가 빌드됩니다.

.NET의 예 `appsettings.json`

```
{
  "XRay": {
    "AWSXRayPlugins": "EC2Plugin"
  }
}
```

.NET Core Program.cs - 레코더 구성의 예

```
using Amazon.XRay.Recorder.Core;
...
AWSXRayRecorder.InitializeInstance(configuration);
```

Tracing configuration with OpenTelemetry SDK

다음 종속성을 추가합니다.

```
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
```

.NET 애플리케이션의 경우 Global TracerProvider를 설정하여 OpenTelemetry SDK를 구성합니다. 다음 예제 구성은에 대한 계측도 활성화합니다`ASP.NET Core`. 를 계측하려면 단원을 `ASP.NET`참조하십시오수신 요청 추적(ASP.NET 및 ASP.NET 코어 계측). OpenTelemetry를 다른 프레임워크와 함께 사용하려면 레지스트리에서 지원되는 프레임워크에 대한 추가 라이브러리를 참조하세요.

다음 구성 요소를 구성하는 것이 좋습니다.

- `An OTLP Exporter` - CloudWatch Agent/OpenTelemetry Collector로 트레이스를 내보내는 데 필요합니다.
- AWS X-Ray 전파기 - 추적 컨텍스트를 AWS X-Ray와 통합된 서비스로 전파하는 데 필요합니다.
- AWS X-Ray 원격 샘플러 - X-Ray 샘플링 규칙을 사용하여 요청을 샘플링해야 하는 경우 필요합니다.
- `Resource Detectors` (예: Amazon EC2 리소스 감지기) - 애플리케이션을 실행하는 호스트의 메타데이터 감지

```
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
```

콘솔 앱에 OpenTelemetry를 사용하려면 프로그램을 시작할 때 다음 OpenTelemetry 구성을 추가합니다.

```
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
```

추적 데이터 수동 생성

With X-Ray SDK

X-Ray SDK를 사용하면 X-Ray 세그먼트 및 하위 세그먼트를 수동으로 생성하는 데 `BeginSegment` 및 `BeginSubsegment` 메서드가 필요했습니다.

```
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

.NET에서는 활동 API를 사용하여 사용자 지정 스팬을 생성하여 계측 라이브러리로 캡처되지 않은 내부 활동의 성능을 모니터링할 수 있습니다. 참고로 서버는 X-Ray 세그먼트로 변환되고 다른 모든 스팬은 X-Ray 하위 세그먼트로 변환됩니다.

필요한 만큼 `ActivitySource` 인스턴스를 생성할 수 있지만 전체 애플리케이션/서비스에 대해 하나만 사용하는 것이 좋습니다.

```
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
    }
}
```

OpenTelemetry SDK를 사용하여 트레이스에 주석 및 메타데이터 추가

활동에서 SetTag 메서드를 사용하여 사용자 지정 키-값 페어를 스팬에 속성으로 추가할 수도 있습니다. 기본적으로 모든 스팬 속성은 X-Ray 원시 데이터의 메타데이터로 변환됩니다. 속성이 메타데이터가 아닌 주석으로 변환되도록 하려면 해당 속성의 키를 aws.xray.annotations 속성 목록에 추가할 수 있습니다.

```
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
    }
}
```

OpenTelemetry 자동 계측 사용

.NET용 OpenTelemetry 자동 계측 솔루션을 사용하고 애플리케이션에서 수동 계측을 수행해야 하는 경우, 예를 들어 자동 계측 라이브러리에서 다루지 않는 섹션의 애플리케이션 자체 내에서 코드를 계측해야 하는 경우.

하나의 글로벌 만 있을 수 있으므로 TracerProvider수동 계측은 자동 계측과 함께 사용하는 TracerProvider 경우 자체 계측을 인스턴스화해서는 안 됩니다. TracerProvider를 사용하는 경우 사용자 지정 수동 추적은 OpenTelemetry SDK를 통해 자동 계측 또는 수동 계측을 사용할 때와 동일한 방식으로 작동합니다.

수신 요청 추적(ASP.NET 및 ASP.NET 코어 계측)

With X-Ray SDK

ASP.NET 애플리케이션에서 제공하는 요청을 계측하려면 global.asax 파일의 Init 메서드RegisterXRay에서를 호출하는 방법에 https://docs.aws.amazon.com/xray/latest/devguide/xray-sdk-dotnet-messagehandler.html 대한 자세한 내용은 섹션을 참조하세요.

```
AWSXRayASPNET.RegisterXRay(this, "MyApp");
```

ASP.NET 코어 애플리케이션에서 제공하는 요청을 계측하기 위해 UseXRay 메서드는 Startup 클래스의 Configure 메서드에 있는 다른 미들웨어보다 먼저 호출됩니다.

```
app.UseXRay("MyApp");
```

With OpenTelemetry SDK

또한 OpenTelemetry는 ASP.NET 및 ASP.NET 코어에 대한 수신 웹 요청에 대한 추적을 수집하는 계측 라이브러리를 제공합니다. 다음 섹션에서는 추적기 공급자를 생성할 때 ASP.NET 또는 ASP.NET 코어 계측을 추가하는 방법을 포함하여 OpenTelemetry 구성에 이러한 라이브러리 계측을 추가하고 활성화하는 데 필요한 단계를 나열합니다.

OpenTelemetry.Instrumentation.AspNet,을 활성화하는 방법에 대한 자세한 내용은 OpenTelemetry.Instrumentation.AspNet을 활성화하는 단계 및 OpenTelemetry.Instrumentation.AspNetCore,를 활성화하는 방법에 대한 자세한 내용은 OpenTelemetry.Instrumentation.AspNetCore를 활성화하는 단계를 참조하세요.

AWS SDK 계측

With X-Ray SDK

를 호출하여 모든 AWS SDK 클라이언트를 설치합니다`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
```

특정 AWS 서비스 클라이언트 계측에는 다음 방법 중 하나를 사용합니다.

```
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

다음 코드 예제의 경우 다음 종속성이 필요합니다.

```
dotnet add package OpenTelemetry.Instrumentation.AWS
```

AWS SDK를 계측하려면 Global TracerProvider가 설정된 OpenTelemetry SDK 구성을 업데이트합니다.

```
builder.Services.AddOpenTelemetry()
    ...
    .WithTracing(tracing => tracing
        .AddAWSInstrumentation()
        ...
```

발신 HTTP 호출 구성

With X-Ray SDK

X-Ray .NET SDK는 확장 메서드를 통해 `GetResponseTraced()` 또는를 사용할 `GetAsyncResponseTraced()` 때 `System.Net.HttpWebRequest`또는를 사용할 때 `HttpClientXRayTracingHandler` 핸들러를 사용하여 발신 HTTP 호출을 추적합니다.`System.Net.Http.HttpClient`.

With OpenTelemetry SDK

다음 코드 예제의 경우 다음 종속성이 필요합니다.

```
dotnet add package OpenTelemetry.Instrumentation.Http
```

`System.Net.Http.HttpClient` 및를 계측하려면 Global TracerProvider가 설정된 OpenTelemetry SDK 구성을 `System.Net.HttpWebRequest`업데이트합니다.

```
builder.Services.AddOpenTelemetry()
    ...
    .WithTracing(tracing => tracing
        .AddHttpClientInstrumentation()
        ...
```

다른 라이브러리에 대한 계측 지원

.NET 계측 라이브러리용 OpenTelemetry 레지스트리를 검색하고 필터링하여 OpenTelemetry가 라이브러리에 대한 계측을 지원하는지 확인할 수 있습니다. 검색을 시작하려면 레지스트리를 참조하세요.

Lambda 계측

With X-Ray SDK

Lambda와 함께 X-Ray SDK를 사용하려면 다음 절차가 필요했습니다.

1. Lambda 함수에서 활성 추적 활성화
2. Lambda 서비스는 핸들러의 호출을 나타내는 세그먼트를 생성합니다.
3. X-Ray SDK를 사용하여 하위 세그먼트 또는 계측 라이브러리 생성

With OpenTelemetry-based solutions

AWS 벤딩된 Lambda 계층으로 Lambda를 자동으로 계측할 수 있습니다. 두 가지 해결 방법이 있습니다.

- (권장) CloudWatch Application Signals lambda 계층
- 성능을 높이려면를 사용하여 Lambda 함수`OpenTelemetry Manual Instrumentation`에 대한 OpenTelemetry 트레이스를 생성하는 것이 좋습니다.

AWS Lambda용 OpenTelemetry 수동 계측

다음은 Lambda 함수 코드(계측 제외) 예제입니다.

```
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);
        }
    }
}
```

Lambda 핸들러와 Amazon S3 클라이언트를 수동으로 계측하려면 다음을 수행합니다.

1. TracerProvider 인스턴스화 - TracerProvider는 `XrayUdpSpanExporter`, ParentBased Always On Sampler 및 Lambda 함수 이름으로 `service.name` 설정된 `Resource` 로 구성하는 것이 좋습니다.
2. 를 호출하여에 SDK 클라이언트 계측`AddAWSInstrumentation()`을 추가하여 OpenTemetry AWS AWS SDK 계측으로 Amazon S3 클라이언트 계측 `TracerProvider`
3. 원래 Lambda 함수와 동일한 서명으로 래퍼 함수를 생성합니다. `AWSLambdaWrapper.Trace()` API를 호출하고 `TracerProvider`, 원래 Lambda 함수 및 해당 입력을 파라미터로 전달합니다. 래퍼 함수를 Lambda 핸들러 입력으로 설정합니다.

다음 코드 예제의 경우 다음 종속성이 필요합니다.

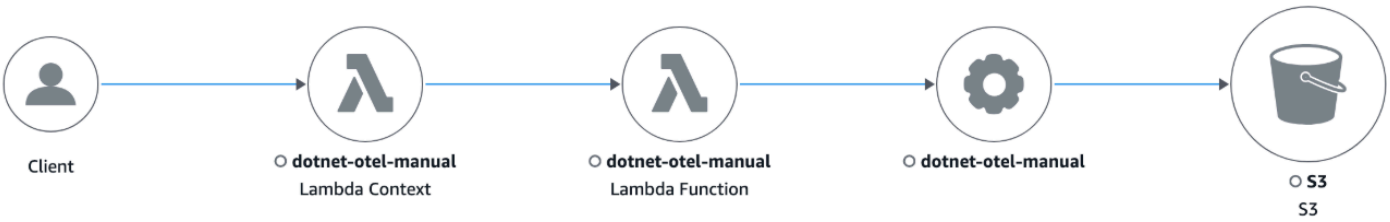
```
dotnet add package OpenTelemetry.Instrumentation.AWSLambda
dotnet add package OpenTelemetry.Instrumentation.AWS
dotnet add package OpenTelemetry.Resources.AWS
dotnet add package AWS.Distro.OpenTelemetry.Exporter.Xray.Udp
```

다음 코드는 필요한 변경 후 Lambda 함수를 보여줍니다. 추가 사용자 지정 스팬을 생성하여 자동으로 제공되는 스팬을 보완할 수 있습니다.

```
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);
        }
    }
}
```

이 Lambda를 호출할 때 CloudWatch 콘솔의 트레이스 맵에 다음 트레이스가 표시됩니다.



OpenTelemetry Python으로 마이그레이션

이 가이드는 Python 애플리케이션을 X-Ray SDK에서 OpenTelemetry 계측으로 마이그레이션하는 데 도움이 됩니다. 일반적인 시나리오에 대한 코드 예제와 함께 자동 및 수동 계측 접근 방식을 모두 다룹니다.

Sections

제로 코드 자동 계측 솔루션

X-Ray SDK를 사용하면 요청을 추적하도록 애플리케이션 코드를 수정해야 했습니다. OpenTelemetry는 요청을 추적하기 위한 제로 코드 자동 계측 솔루션을 제공합니다. OpenTelemetry를 사용하면 제로 코드 자동 계측 솔루션을 사용하여 요청을 추적할 수 있습니다.

OpenTelemetry 기반 자동 계측을 사용하는 제로 코드

1. Python용 AWS Distro for OpenTelemetry(ADOT) 자동 계측 사용 - Python 애플리케이션에 대한 자동 계측은 AWS Distro for OpenTelemetry Python 자동 계측을 사용한 추적 및 지표를 참조하세요.

   (선택 사항) ADOT Python 자동 계측 AWS 을 사용하여에서 애플리케이션을 자동으로 계측할 때 CloudWatch Application Signals를 활성화하여 현재 애플리케이션 상태를 모니터링하고 비즈니스 목표에 따라 장기 애플리케이션 성능을 추적할 수도 있습니다. Application Signals는 애플리케이션, 서비스 및 종속성에 대한 통합 애플리케이션 중심 보기를 제공하고 애플리케이션 상태를 모니터링하고 분류하는 데 도움이 됩니다.

2. OpenTelemetry Python 제로 코드 자동 계측 사용 - OpenTelemetry Python을 사용한 자동 계측은 Python 제로 코드 계측을 참조하세요.

애플리케이션을 수동으로 계측

`pip` 명령을 사용하여 애플리케이션을 수동으로 계측할 수 있습니다.

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
```

추적 설정 초기화

With X-Ray SDK

X-Ray에서는 전역`xray_recorder`이 초기화되고 이를 사용하여 세그먼트 및 하위 세그먼트를 생성합니다.

With OpenTelemetry SDK

> **Note**
>
> X-Ray 원격 샘플링은 현재 OpenTelemetry Python에 대해 구성할 수 없습니다. 그러나 X-Ray 원격 샘플링에 대한 지원은 현재 Python용 ADOT 자동 계측을 통해 제공됩니다.

OpenTelemetry에서는 글로벌를 초기화해야 합니다`TracerProvider`. 이를 사용하면 애플리케이션의 모든 위치에서 스팬을 생성하는 데 사용할 수 있는 추적기를 얻을 `TracerProvider`수 있습니다. 다음 구성 요소를 구성하는 것이 좋습니다.

- `OTLPSpanExporter` - CloudWatch Agent/OpenTelemetry Collector로 트레이스를 내보내는 데 필요합니다.
- AWS X-Ray 전파기 - 추적 컨텍스트를 X-Ray와 통합된 AWS 서비스로 전파하는 데 필요합니다.

```
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
```

Python용 ADOT 자동 계측 사용

Python용 ADOT 자동 계측을 사용하여 Python 애플리케이션에 대해 OpenTelemetry를 자동으로 구성할 수 있습니다. ADOT 자동 계측을 사용하면 수신 요청을 추적하거나 AWS SDK 또는 HTTP 클라이언트와 같은 라이브러리를 추적하기 위해 수동으로 코드를 변경할 필요가 없습니다. 자세한 내용은 AWS Distro for OpenTelemetry Python Auto-Instrumentation을 사용한 추적 및 지표를 참조하세요.

Python용 ADOT 자동 계측은 다음을 지원합니다.

- 환경 변수를 통한 X-Ray 원격 샘플링 `export OTEL_TRACES_SAMPLER=xray`
- X-Ray 추적 컨텍스트 전파(기본적으로 활성화됨)
- 리소스 감지(Amazon EC2, Amazon ECS 및 Amazon EKS 환경에 대한 리소스 감지는 기본적으로 활성화되어 있음)
- 지원되는 모든 OpenTelemetry 계측에 대한 자동 라이브러리 계측은 기본적으로 활성화됩니다. `OTEL_PYTHON_DISABLED_INSTRUMENTATIONS` 환경 변수를 통해 선택적으로를 비활성화할 수 있습니다(모두 기본적으로 활성화되어 있음).
- 스팬 수동 생성

X-Ray 서비스 플러그인에서 OpenTelemetry AWS 리소스 공급자로

X-Ray SDK는 Amazon EC2, Amazon ECS 및 Elastic Beanstalk와 같은 호스팅 서비스에서 플랫폼별 정보를 캡처`xray_recorder`하기 위해에 추가할 수 있는 플러그인을 제공합니다. 이는 정보를 리소스 속성으로 캡처하는 OpenTelemetry의 리소스 공급자와 유사합니다. 다양한 AWS 플랫폼에서 사용할 수 있는 리소스 공급자가 여러 개 있습니다.

- AWS 확장 패키지를 설치하여 시작합니다. `pip install opentelemetry-sdk-extension-aws`
- 원하는 리소스 감지기를 구성합니다. 다음 예제는 OpenTelemetry SDK에서 Amazon EC2 리소스 공급자를 구성하는 방법을 보여줍니다.

```
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)
```

수신 요청 추적

With X-Ray SDK

X-Ray Python SDK는 Django, Flask 및 Bottle과 같은 애플리케이션 프레임워크를 지원하여 실행 중인 Python 애플리케이션에 대한 수신 요청을 추적합니다. 이는 각 프레임워크의 애플리케이션에 `XRayMiddleware`를 추가하여 수행됩니다.

With OpenTelemetry SDK

OpenTelemetry는 특정 계측 라이브러리를 통해 Django 및 Flask에 대한 계측을 제공합니다. OpenTelemetry에서 사용할 수 있는 Bottle에 대한 계측은 없으며 OpenTelemetry WSGI 계측 을 사용하여 애플리케이션을 추적할 수 있습니다.

다음 코드 예제의 경우 다음 종속성이 필요합니다.

```
pip install opentelemetry-instrumentation-flask
```

애플리케이션 프레임워크에 계측을 추가하기 전에 OpenTelemetry SDK를 초기화하고 글로벌 TracerProvider를 등록해야 합니다. 그렇지 않으면 추적 작업은가 됩니다no-ops. 글로벌를 구성한 후에는 애플리케이션 프레임워크에 계측기를 사용할 `TracerProvider`수 있습니다. 다음 예제에서는 Flask 애플리케이션을 보여줍니다.

```
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 계측

With X-Ray SDK

X-Ray Python SDK는 `botocore` 라이브러리를 패치하여 AWS SDK 클라이언트 요청을 추적합니다. 자세한 내용은 Python용 X-Ray AWS SDK를 사용하여 SDK 호출 추적을 참조하세요. 애플리케

이션에서 `patch_all()` 메서드는를 사용하여 또는 라이브러리를 사용하여 모든 `boto3`라이브러리 `botocore` 또는 패치를 선택적으로 계측하는 데 사용됩니다`patch((['botocore']))`. 선택한 메서드는 애플리케이션의 모든 Boto3 클라이언트를 계측하고 이러한 클라이언트를 사용하여 이루어진 모든 호출에 대해 하위 세그먼트를 생성합니다.

With OpenTelemetry SDK

다음 코드 예제의 경우 다음 종속성이 필요합니다.

```
pip install opentelemetry-instrumentation-botocore
```

프로그래밍 방식으로 OpenTelemetry Botocore 계측을 사용하여 애플리케이션의 모든 Boto3 클라이언트를 계측합니다. 다음 예제에서는 `botocore` 계측을 보여줍니다.

```
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}")
```

요청을 통해 발신 HTTP 호출 계측

With X-Ray SDK

X-Ray Python SDK는 요청 라이브러리를 패치하여 요청을 통해 발신 HTTP 호출을 추적합니다. 자세한 내용은 Python용 X-Ray SDK를 사용하여 다운스트림 HTTP 웹 서비스에 대한 호출 추적을 참조하세요. 애플리케이션에서 `patch_all()` 메서드를 사용하여 모든 라이브러리를 계측하거나를 사용하여 요청 라이브러리를 선택적으로 패치할 수 있습니다`patch((['requests']))`. 모든 옵션은 `requests` 라이브러리를 계측하여를 통해 이루어진 모든 호출에 대해 하위 세그먼트를 생성합니다`requests`.

With OpenTelemetry SDK

다음 코드 예제의 경우 다음 종속성이 필요합니다.

```
pip install opentelemetry-instrumentation-requests
```

프로그래밍 방식으로 OpenTelemetry Requests Instrumentation을 사용하여 요청 라이브러리를 계측하여 애플리케이션에서 수행한 HTTP 요청에 대한 트레이스를 생성합니다. 자세한 내용은 OpenTelemetry 요청 계측을 참조하세요. 다음 예제에서는 `requests` 라이브러리 계측을 보여줍니다.

```
from opentelemetry.instrumentation.requests import RequestsInstrumentor

# Instrument Requests
RequestsInstrumentor().instrument()

...

    example_session = requests.Session()
    example_session.get(url="https://example.com")
```

또는 기본 `urllib3` 라이브러리를 계측하여 HTTP 요청을 추적할 수도 있습니다.

```
# 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")
```

다른 라이브러리에 대한 계측 지원

지원되는 라이브러리, 프레임워크, 애플리케이션 서버 및 JVM에서 OpenTelemetry Python에 지원되는 라이브러리 계측의 전체 목록을 찾을 수 있습니다. JVMs

또는 OpenTelemetry 레지스트리를 검색하여 OpenTelemetry가 계측을 지원하는지 확인할 수 있습니다. 검색을 시작하려면 레지스트리를 참조하세요.

추적 데이터 수동 생성

Python 애플리케이션에서를 사용하여 세그먼트 및 하위 세그먼트를 생성할 수 `xray_recorder` 있습니다. 자세한 내용은 Python 코드 수동 계측을 참조하세요. 트레이스 데이터에 주석과 메타데이터를 수동으로 추가할 수도 있습니다.

OpenTelemetry SDK를 사용하여 스팬 생성

start_as_current_span API를 사용하여 스팬을 시작하고 스팬 생성을 위해 스팬을 설정합니다. 스팬 생성에 대한 예제는 스팬 생성을 참조하세요. 스팬이 시작되고 현재 범위에 있으면 속성, 이벤트, 예외, 링크 등을 추가하여 더 많은 정보를 추가할 수 있습니다. X-Ray에 세그먼트와 하위 세그먼트가 있는 방식과 마찬가지로 OpenTelemetry에는 다양한 종류의 스팬이 있습니다. SERVER 종류 범위만 X-Ray 세그먼트로 변환되고 나머지는 X-Ray 하위 세그먼트로 변환됩니다.

```
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
```

OpenTelemetry SDK를 사용하여 트레이스에 주석 및 메타데이터 추가

X-Ray Python SDK는 트레이스에 주석put_annotation과 메타데이터를 추가하기 put_metadata 위한 별도의 APIs와를 제공합니다. OpenTelemetry SDK에서 주석과 메타데이터는 단순히 스팬의 속성이며 set_attribute API를 통해 추가됩니다.

트레이스의 주석으로 사용할 범위 속성은 값이 주석의 키-값 페어 목록aws.xray.annotations인 예약된 키 아래에 추가됩니다. 다른 모든 스팬 속성은 변환된 세그먼트 또는 하위 세그먼트의 메타데이터가 됩니다.

또한 ADOT 수집기를 사용하는 경우 수집기 구성indexed_attributes에서를 지정하여 X-Ray 주석으로 변환해야 하는 스팬 속성을 구성할 수 있습니다.

아래 예제에서는 OpenTelemetry SDK를 사용하여 트레이스에 주석과 메타데이터를 추가하는 방법을 보여줍니다.

```
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 계측

X-Ray에서 Lambda 함수를 모니터링하려면 X-Ray를 활성화하고 함수 호출 역할에 적절한 권한을 추가해야 합니다. 또한 함수의 다운스트림 요청을 추적하는 경우 X-Ray Python SDK로 코드를 계측합니다.

OpenTelemetry for X-Ray에서는 Application Signals가 꺼진 상태에서 CloudWatch Application Signals lambda 계층을 사용하는 것이 좋습니다. 이렇게 하면 함수가 자동으로 계측되고 함수 호출 및 함수의 다운스트림 요청에 대한 스팬이 생성됩니다. 추적 외에도 Application Signals를 사용하여 함수의 상태를 모니터링하는 데 관심이 있는 경우 Lambda에서 애플리케이션 활성화를 참조하세요.

- AWS OpenTelemetry ARN용 Lambda 계층에서 함수에 필요한 Lambda 계층 ARNs 찾아 추가합니다.
- 함수에 대해 다음 환경 변수를 설정합니다.
  - AWS_LAMBDA_EXEC_WRAPPER=/opt/otel-instrument - 함수에 대한 자동 계측을 로드합니다.
  - OTEL_AWS_APPLICATION_SIGNALS_ENABLED=false - Application Signals 모니터링을 비활성화합니다.

Lambda 계측을 사용하여 수동으로 스팬 생성

또한 함수 내에서 사용자 지정 스팬을 생성하여 작업을 추적할 수 있습니다. Application Signals Lambda 계층 자동 계측과 함께 `opentelemetry-api` 패키지만 사용하면 됩니다.

1. 함수에를 종속성`opentelemetry-api`으로 포함
2. 다음 코드 조각은 사용자 지정 스팬을 생성하는 샘플입니다.

```
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
    }
```

OpenTelemetry Ruby로 마이그레이션

Ruby 애플리케이션을 X-Ray SDK에서 OpenTelemetry 계측으로 마이그레이션하려면 수동 계측에 대한 다음 코드 예제와 지침을 사용합니다.

Sections

- 다른 라이브러리에 대한 계측 지원
- 추적 데이터 수동 생성
- Lambda 수동 계측

SDK를 사용하여 수동으로 솔루션 계측

Tracing setup with X-Ray SDK

Ruby용 X-Ray SDK를 사용하려면 서비스 플러그인으로 코드를 구성해야 했습니다.

```
require 'aws-xray-sdk'

XRay.recorder.configure(plugins: [:ec2, :elastic_beanstalk])
```

Tracing setup with OpenTelemetry SDK

> **Note**
>
> X-Ray 원격 샘플링은 현재 OpenTelemetry Ruby에 대해 구성할 수 없습니다.

Ruby on Rails 애플리케이션의 경우 구성 코드를 Rails 이니셜라이저에 배치합니다. 자세한 내용은 시작하기를 참조하십시오. 수동으로 구성된 모든 Ruby 프로그램의 경우 `OpenTelemetry::SDK.configure` 메서드를 사용하여 OpenTelemetry Ruby SDK를 구성해야 합니다.

먼저 다음 패키지를 설치합니다.

```
bundle add opentelemetry-sdk opentelemetry-exporter-otlp opentelemetry-propagator-
xray
```

그런 다음 프로그램이 초기화될 때 실행되는 구성 코드를 통해 OpenTelemetry SDK를 구성합니다. 다음 구성 요소를 구성하는 것이 좋습니다.

- `OTLP Exporter` - CloudWatch 에이전트 및 OpenTelemetry 수집기로 트레이스를 내보내는 데 필요합니다.

- An AWS X-Ray Propagator - 추적 컨텍스트를 X-Ray와 통합된 AWS 서비스로 전파하는 데 필요합니다.

```
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 SDKs도 있습니다. 이를 활성화하면 AWS SDK와 같은 라이브러리에 대한 스팬이 자동으로 생성됩니다. OpenTelemetry는 모든 라이브러리 계측을 활성화하거나 활성화할 라이브러리 계측을 지정하는 옵션을 제공합니다.

모든 계측을 활성화하려면 먼저 `opentelemetry-instrumentation-all` 패키지를 설치합니다.

```
bundle add opentelemetry-instrumentation-all
```

그런 다음 아래와 같이 구성을 업데이트하여 모든 라이브러리 계측을 활성화합니다.

```
require 'opentelemetry/instrumentation/all'
...

OpenTelemetry::SDK.configure do |c|
   ...

  c.use_all() # Enable all instrumentations
end
```

OpenTelemetry SDKs도 있습니다. 이를 활성화하면 AWS SDK와 같은 라이브러리에 대한 스팬이 자동으로 생성됩니다. OpenTelemetry는 모든 라이브러리 계측을 활성화하거나 활성화할 라이브러리 계측을 지정하는 옵션을 제공합니다.

모든 계측을 활성화하려면 먼저 `opentelemetry-instrumentation-all` 패키지를 설치합니다.

```
bundle add opentelemetry-instrumentation-all
```

그런 다음 아래와 같이 구성을 업데이트하여 모든 라이브러리 계측을 활성화합니다.

```
require 'opentelemetry/instrumentation/all'
...

OpenTelemetry::SDK.configure do |c|
   ...

  c.use_all() # Enable all instrumentations
end
```

수신 요청 추적(레일 계측)

With X-Ray SDK

X-Ray SDK를 사용하면 초기화 시 Rails 프레임워크에 맞게 X-Ray 추적이 구성됩니다.

예 - config/initializers/aws_xray.rb

```
Rails.application.config.xray = {
  name: 'my app',
  patch: %I[net_http aws_sdk],
  active_record: true
}
```

With OpenTelemetry SDK

먼저 다음 패키지를 설치합니다.

```
bundle add opentelemetry-instrumentation-rack opentelemetry-instrumentation-
rails opentelemetry-instrumentation-action_pack opentelemetry-instrumentation-
active_record opentelemetry-instrumentation-action_view
```

그런 다음 아래와 같이 구성을 업데이트하여 Rails 애플리케이션에 대한 계측을 활성화합니다.

```
# 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 계측

With X-Ray SDK

AWS SDK의 발신 AWS 요청을 계측하기 위해 AWS SDK 클라이언트는 다음 예제와 같이 X-Ray로 패치됩니다.

```
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 SDK for Ruby V3는 OpenTelemetry 추적을 기록하고 내보낼 수 있도록 지원합니다. 서비스 클라이언트에 대해 OpenTelemetry를 구성하는 방법에 대한 자세한 내용은 AWS SDK for Ruby의 관찰성 기능 구성을 참조하세요.

발신 HTTP 호출 구성

외부 서비스에 대한 HTTP 호출을 수행할 때 자동 계측을 사용할 수 없거나 충분한 세부 정보를 제공하지 않는 경우 호출을 수동으로 계측해야 할 수 있습니다.

With X-Ray SDK

다운스트림 호출을 계측하기 위해 Ruby용 X-Ray SDK를 사용하여 애플리케이션이 사용하는 `net/http` 라이브러리를 패치했습니다.

```
require 'aws-xray-sdk'

config = {
  name: 'my app',
  patch: %I[net_http]
}

XRay.recorder.configure(config)
```

With OpenTelemetry SDK

OpenTelemetry를 사용하여 `net/http` 계측을 활성화하려면 먼저 `opentelemetry-instrumentation-net_http` 패키지를 설치합니다.

```
bundle add opentelemetry-instrumentation-net_http
```

그런 다음 아래와 같이 구성을 업데이트하여 net/http 계측을 활성화합니다.

```
OpenTelemetry::SDK.configure do |c|
   ...

  c.use 'OpenTelemetry::Instrumentation::Net::HTTP'
  ...

end
```

다른 라이브러리에 대한 계측 지원

OpenTelemetry Ruby에 지원되는 라이브러리 계측의 전체 목록은 opentelemetry-ruby-contrib에서 확인할 수 있습니다.

또는 OpenTelemetry 레지스트리를 검색하여 OpenTelemetry가 계측을 지원하는지 확인할 수 있습니다. 자세한 내용은 레지스트리를 참조하세요.

추적 데이터 수동 생성

With X-Ray SDK

X-Ray를 사용하는 aws-xray-sdk 패키지에서는 애플리케이션을 추적하기 위해 세그먼트와 하위 하위 세그먼트를 수동으로 생성해야 했습니다. 세그먼트 또는 하위 세그먼트에 X-Ray 주석 및 메타데이터를 추가했을 수도 있습니다.

```
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

사용자 지정 스팬을 사용하여 계측 라이브러리로 캡처되지 않은 내부 활동의 성능을 모니터링할 수 있습니다. 종류의 스팬 서버만 X-Ray 세그먼트로 변환되고 다른 모든 스팬은 X-Ray 하위 세그먼트로 변환됩니다. 기본적으로 스팬은 입니다`INTERNAL`.

먼저 Tracer를 생성하여 `OpenTelemetry.tracer_provider.tracer('<YOUR_TRACER_NAME>')` 메서드를 통해 얻을 수 있는 스팬을 생성합니다. 그러면 애플리케이션의 OpenTelemetry 구성에 전역적으로 등록된 Tracer 인스턴스가 제공됩니다. 전체 애플리케이션에 대해 단일 Tracer를 사용하는 것이 일반적입니다. OpenTelemetry 추적기를 생성하고 이를 사용하여 스팬을 생성합니다.

```
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
```

OpenTelemetry SDK를 사용하여 트레이스에 주석 및 메타데이터 추가

`set_attribute` 메서드를 사용하여 각 스팬에 속성을 추가합니다. 기본적으로 이러한 모든 스팬 속성은 X-Ray 원시 데이터의 메타데이터로 변환됩니다. 속성이 메타데이터가 아닌 주석으로 변환되도록 하려면 해당 속성 키를 `aws.xray.annotations` 속성 목록에 추가할 수 있습니다. 자세한 내용은 사용자 지정 X-Ray 주석 활성화를 참조하세요.

```
# 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 수동 계측

With X-Ray SDK

Lambda에서 활성 추적이 활성화된 후에는 X-Ray SDK를 사용하는 데 추가 구성이 필요하지 않습니다. Lambda는 Lambda 핸들러 호출을 나타내는 세그먼트를 생성하고 추가 구성 없이 X-Ray SDK를 사용하여 하위 세그먼트 또는 계측 라이브러리를 생성할 수 있습니다.

With OpenTelemetry SDK

다음 샘플 Lambda 함수 코드(계측 제외)를 고려합니다.

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

Lambda를 수동으로 계측하려면 다음을 수행해야 합니다.

1. Lambda에 대해 다음 Gem을 추가합니다.

   ```
   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. Lambda 핸들러 외부에서 OpenTelemetry SDK를 초기화합니다. OpenTelemetry SDK는 다음과 같이 구성하는 것이 좋습니다.

   1. 트레이스를 Lambda의 UDP X-Ray 엔드포인트로 전송하기 위한 X-Ray UDP 스팬 내보내기 기능이 있는 간단한 스팬 프로세서
   2. X-Ray Lambda 전파기

3. `service_name` Lambda 함수 이름으로 설정할 구성

3. Lambda 핸들러 클래스에서 다음 줄을 추가하여 Lambda 핸들러를 계측합니다.

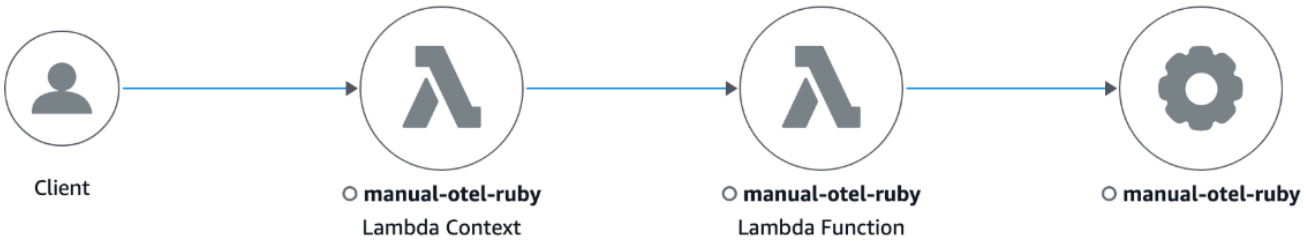
```
  class Handler
        extend OpenTelemetry::Instrumentation::AwsLambda::Wrap
        ...

        instrument_handler :process
    end
```

다음 코드는 필요한 변경 후 Lambda 함수를 보여줍니다. 추가 사용자 지정 스팬을 생성하여 자동으로 제공되는 스팬을 보완할 수 있습니다.

```
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
```

다음은 Ruby로 작성된 계측된 Lambda 함수의 트레이스 맵 예제입니다.



Lambda 계층을 사용하여 Lambda에 대해 OpenTelemetry를 구성할 수도 있습니다. 자세한 내용은 OpenTelemetry AWS-Lambda 계측을 참조하세요.

AWS CloudFormation로 X-Ray 리소스 만들기

AWS X-Ray 는 AWS 리소스 및 인프라를 생성하고 관리하는 데 소요되는 시간을 줄일 수 있도록 리소스를 모델링하고 설정하는 데 도움이 되는 AWS CloudFormation서비스인와 통합됩니다. 원하는 모든 AWS 리소스를 설명하는 템플릿을 생성하면가 해당 리소스를 AWS CloudFormation 프로비저닝하고 구성합니다.

를 사용하면 템플릿을 재사용하여 X-Ray 리소스를 일관되고 반복적으로 설정할 AWS CloudFormation 수 있습니다. 리소스를 한 번 설명한 다음 여러 AWS 계정 및 리전에서 동일한 리소스를 반복적으로 프로비저닝합니다.

X-Ray 및 AWS CloudFormation 템플릿

X-Ray 및 관련 서비스에 대한 리소스를 프로비저닝하고 구성하려면 AWS CloudFormation 템플릿을 이해해야 합니다. 템플릿은 JSON 또는 YAML로 서식 지정된 텍스트 파일입니다. 이러한 템플릿은 AWS CloudFormation 스택에서 프로비저닝하려는 리소스를 설명합니다. JSON 또는 YAML에 익숙하지 않은 경우 Designer를 사용하여 AWS CloudFormation AWS CloudFormation 템플릿을 시작할 수 있습니다. 자세한 내용은 AWS CloudFormation 사용 설명서에서 AWS CloudFormation Designer이란 무엇입니까?를 참조하세요.

X-Ray는에서 `AWS::XRay::Group`, `AWS::XRay::SamplingRule`및 `AWS::XRay::ResourcePolicy` 리소스 생성을 지원합니다 AWS CloudFormation. JSON 및 YAML 템플릿의 예를 비롯한 자세한 내용은 AWS CloudFormation 사용 설명서의 X-Ray 리소스 유형 참조를 참조하세요.

에 대해 자세히 알아보기 AWS CloudFormation

에 대해 자세히 알아보려면 다음 리소스를 AWS CloudFormation참조하세요.

- AWS CloudFormation
- AWS CloudFormation 사용 설명서
- AWS CloudFormation API Reference
- AWS CloudFormation 명령줄 인터페이스 사용 설명서

X-Ray 샘플링 규칙 및 그룹 태그 지정하기

태그는 AWS 리소스를 식별하고 구성하는 데 사용할 수 있는 단어 또는 구문입니다. 각 리소스에 여러 개의 태그를 추가할 수 있습니다. 각 태그는 사용자가 정의하는 키와 선택적 값이 포함됩니다. 예를 들어, 태그 키는 **domain**이고 태그 값은 **example.com**일 수 있습니다. 추가하는 태그를 기준으로 리소스를 검색하고 필터링할 수 있습니다. 태그에 대한 자세한 내용은 AWS 일반 참조 안내서의 AWS 리소스 태깅을 참조하세요.

태그를 사용하여 CloudFront 배포에 대한 태그 기반 권한을 적용할 수 있습니다. 자세한 내용은 리소스 태그를 사용한 AWS 리소스 액세스 제어를 참조하세요.

> **Note**
>
> Tag Editor 및 AWS Resource Groups는 현재 X-Ray 리소스를 지원하지 않습니다. AWS X-Ray 콘솔 또는 API를 사용하여 태그를 추가하고 관리합니다.

X-Ray 콘솔, API, AWS CLI SDK SDKs AWS Tools for Windows PowerShell. 자세한 내용은 다음 설명서를 참조하세요.

- X-Ray API – AWS X-Ray API 참조 자료에서 다음 작업을 참조하십시오.
  - ListTagsForResource
  - CreateSamplingRule
  - CreateGroup
  - TagResource
  - UntagResource
- AWS CLI - AWS CLI 명령 참조의 xray 참조
- SDK - AWS 설명서 페이지에서 해당 SDK 설명서 참조

> **Note**
>
> X-Ray 리소스에 태그를 추가하거나 변경할 수 없거나 특정 태그가 있는 리소스를 추가할 수 없다면 이 작업을 수행할 수 있는 권한이 없는 것입니다. 액세스를 요청하려면 X-Ray에서 관리자 권한이 있는 엔터프라이즈의 AWS 사용자에게 문의하세요.

주제

태그 제한

태그에 적용되는 제한은 다음과 같습니다.

- 리소스당 최대 태그 수 – 50개
- 최대 키 길이 - 유니코드 128자
- 최대 값 길이 - 유니코드 256자
- 키 및 값의 유효값 - a-z, A-Z, 0-9, 공백 및 특수 문자 _ . : / = + - 및 @
- 태그 키와 값은 대소문자를 구분합니다.
- 키 접두사로 `aws:`를 사용하지 마세요. AWS 전용입니다.

> **Note**
>
> 시스템 태그는 수정하거나 삭제할 수 없습니다.

콘솔에서 태그 관리

엑스레이 그룹 또는 샘플링 규칙을 만들 때 태그(선택 사항)를 추가할 수 있습니다. 태그는 나중에 콘솔에서 변경하거나 삭제할 수 있습니다.

다음 절차는 X-Ray 콘솔에서 그룹 및 샘플링 규칙에 대한 태그를 추가, 편집 및 삭제하는 방법을 설명합니다.

주제

- 그룹 태그 편집 또는 삭제하기 (콘솔)
- 샘플링 규칙 태그 편집 또는 삭제하기 (콘솔)

새 그룹에 태그 추가 (콘솔)

새 X-Ray 그룹을 생성할 때 그룹 생성 페이지에서 태그(선택 사항)를 추가할 수 있습니다.

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/xray/home://https://https://https://https:///:https:///:///:///https://www.
2. 탐색 창에서 구성을 확장하고 그룹을 선택합니다.
3. 그룹 생성을 선택합니다.
4. 그룹 생성 페이지에서 그룹의 이름과 필터 표현식을 지정합니다. 이러한 속성에 대한 자세한 내용은 그룹 구성 단원을 참조하십시오.
5. 태그에서 태그 키와 태그 값(선택 사항)을 입력합니다. 예를 들어, 태그 키 **Stage**와 태그 값 **Production**을 입력하여 이 그룹이 프로덕션용임을 나타낼 수 있습니다. 태그를 추가할 때 새 줄이 표시되어 필요한 경우 다른 태그를 추가할 수 있습니다. 태그에 대한 제한 사항은 태그 제한을 참조하십시오.
6. 태그 추가가 완료되면 그룹 생성을 클릭합니다.

새 샘플링 규칙에 태그 추가하기 (콘솔)

새 X-Ray 샘플링 규칙을 생성할 때 샘플링 규칙 생성 페이지에서 태그를 추가할 수 있습니다.

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/xray/home://https://https://https:///:https:///:///:///https://www.
2. 탐색 창에서 구성을 확장하고 샘플링을 선택합니다.
3. 샘플링 규칙 생성을 선택합니다.
4. 샘플링 규칙 생성 페이지에서 이름, 우선 순위, 제한, 매칭 기준 및 매칭 속성을 지정합니다. 이러한 속성에 대한 자세한 내용은 샘플링 규칙 구성 단원을 참조하십시오.
5. 태그에서 태그 키와 태그 값(선택 사항)을 입력합니다. 예를 들어, 태그 키 **Stage**와 태그 값 **Production**을 입력하여 이 샘플링 규칙이 프로덕션용임을 나타낼 수 있습니다. 태그를 추가할 때 새 줄이 표시되어 필요한 경우 다른 태그를 추가할 수 있습니다. 태그에 대한 제한 사항은 태그 제한을 참조하십시오.
6. 태그 추가가 완료되면 샘플링 규칙 생성을 클릭합니다.

그룹 태그 편집 또는 삭제하기 (콘솔)

그룹 편집 페이지에서 X-Ray 그룹의 태그를 변경하거나 삭제할 수 있습니다.

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/xray/home://https://https://https://https://://https://://https://://://https://www.
2. 탐색 창에서 구성을 확장하고 그룹을 선택합니다.
3. 그룹 테이블에서 그룹의 이름을 선택합니다.
4. 그룹 편집 페이지의 태그에서 태그 키와 값을 편집합니다. 태그 키는 중복해서 사용할 수 없습니다. 태그 값은 선택 사항이며 원하는 경우 값을 삭제할 수 있습니다. 그룹 편집 페이지의 다른 속성에 대한 자세한 내용은 그룹 구성을 참조하십시오. 태그에 대한 제한 사항은 태그 제한을 참조하십시오.
5. 태그를 삭제하려면 태그 오른쪽에 있는 X를 선택합니다.
6. 태그 편집 또는 삭제를 마치면 그룹 업데이트를 선택합니다.

샘플링 규칙 태그 편집 또는 삭제하기 (콘솔)

샘플링 규칙 편집 페이지에서 X-Ray 샘플링 규칙의 태그를 변경하거나 삭제할 수 있습니다.

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/xray/home://https://https://https://https://://https://://https://://://https://www.
2. 탐색 창에서 구성을 확장하고 샘플링을 선택합니다.
3. 샘플링 규칙 테이블에서 샘플링 규칙의 이름을 선택합니다.
4. 태그에서 태그 키 및 값을 편집합니다. 태그 키는 중복해서 사용할 수 없습니다. 태그 값은 선택 사항이며 원하는 경우 값을 삭제할 수 있습니다. 샘플링 규칙 편집 페이지의 다른 속성에 대한 자세한 내용은 샘플링 규칙 구성을 참조하십시오. 태그에 대한 제한 사항은 태그 제한을 참조하십시오.
5. 태그를 삭제하려면 태그 오른쪽에 있는 X를 선택합니다.
6. 태그 편집 또는 삭제를 마치면 샘플링 규칙 업데이트를 선택합니다.

에서 태그 관리 AWS CLI

X-Ray 그룹 또는 샘플링 규칙을 만들 때 태그를 추가할 수 있습니다. AWS CLI 를 사용하여 태그를 생성하고 관리할 수도 있습니다. 기존 그룹 또는 샘플링 규칙에서 태그를 업데이트하려면 AWS X-Ray 콘솔 또는 TagResource 또는 UntagResource APIs 사용합니다.

주제

새 X-Ray 그룹 또는 샘플링 규칙(CLI)에 태그 추가하기

새 X-Ray 그룹 또는 샘플링 규칙을 만들 때 선택적 태그를 추가하려면 다음 명령 중 하나를 사용하세요.

- 새 그룹에 태그를 추가하려면 다음 명령을 실행하여 *group_name*을 그룹 이름으로, *mydomain.com*을 서비스 엔드포인트로, *key_name*을 태그 키로, *#*(선택 사항)을 태그 값으로 대체합니다. 그룹 생성 방법에 대한 자세한 내용은 Groups 섹션을 참조하십시오.

  ```
  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"}]
  ```

  다음은 예입니다.

  ```
  aws xray create-group \
     --group-name "AdminGroup" \
     --filter-expression "service(\"mydomain.com\") {fault OR error}" \
     --tags [{"Key": "Stage","Value": "Prod"},{"Key": "Department","Value": "QA"}]
  ```

- 새 샘플링 규칙에 태그를 추가하려면 다음 명령을 실행하여 *key_name*을 태그 키로 바꾸고, 선택 사항으로 *value*를 태그 값으로 바꿉니다. 이 명령은 `--sampling-rule` 매개변수의 값을 JSON 파일로 지정합니다. 샘플링 규칙 생성 방법에 대한 자세한 내용은 샘플링 규칙 섹션을 참조하십시오.

  ```
  aws xray create-sampling-rule \
     --cli-input-json file://file_name.json
  ```

  다음은 `--cli-input-json` 매개변수로 지정된 JSON 파일 *file_name.json*의 내용입니다.

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

다음 명령은 예제입니다.

```
aws xray create-sampling-rule \
   --cli-input-json file://9000-base-scorekeep.json
```

다음은 `--cli-input-json` 매개변수로 지정된 예제 `9000-base-scorekeep.json` 파일의 내용입니다.

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

기존 리소스에 태그 추가 (CLI)

`tag-resource` 명령을 실행하여 기존 X-Ray 그룹 또는 샘플링 규칙에 태그를 추가할 수 있습니다. 이 방법은 `update-group` 또는 `update-sampling-rule`를 실행하여 태그를 추가하는 것보다 간단합니다.

그룹 또는 샘플링 규칙에 태그를 추가하려면 다음 명령을 실행하여 ARN을 리소스의 ARN으로 바꾸고 추가하려는 태그의 키와 선택적 값을 지정합니다.

```
aws xray tag-resource \
   --resource-arn "ARN" \
   --tag-keys [{"Key":"key_name","Value":"value"}, {"Key":"key_name","Value":"value"}]
```

다음은 예입니다.

```
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"}]
```

리소스의 태그 나열하기 (CLI)

`list-tags-for-resource` 명령을 실행하여 X-Ray 그룹 또는 샘플링 규칙의 태그를 나열할 수 있습니다.

그룹 또는 샘플링 규칙과 연결된 태그를 나열하려면 다음 명령을 실행하여 ARN을 리소스의 ARN으로 바꿉니다.

```
aws xray list-tags-for-resource \
   --resource-arn "ARN"
```

다음은 예입니다.

```
aws xray list-tags-for-resource \
   --resource-arn "arn:aws:xray:us-east-2:01234567890:group/AdminGroup"
```

리소스에서 태그 삭제하기 (CLI)

`untag-resource` 명령을 실행하여 X-Ray 그룹 또는 샘플링 규칙의 태그를 삭제할 수 있습니다.

그룹 또는 샘플링 규칙에 태그를 추가하려면 다음 명령을 실행하여 ARN을 리소스의 ARN으로 바꾸고 삭제하려는 태그의 키를 지정합니다.

`untag-resource` 명령으로 전체 태그만 제거할 수 있습니다. 태그 값을 제거하려면 X-Ray 콘솔을 사용하거나, 태그를 삭제하고 동일한 키이지만 값이 다르거나 비어 있는 새 태그를 추가합니다.

```
aws xray untag-resource \
   --resource-arn "ARN" \
   --tag-keys ["key_name","key_name"]
```

다음은 예입니다.

```
aws xray untag-resource \
   --resource-arn "arn:aws:xray:us-east-2:01234567890:group/group_name" \
   --tag-keys ["Stage","Department"]
```

태그를 기반으로 X-Ray 리소스에 대한 액세스 제어

X-Ray 그룹이나 샘플링 규칙에 태그를 첨부하거나 요청에 포함된 태그를 X-Ray에 전달할 수 있습니다. 태그에 근거하여 액세스를 제어하려면 `xray:ResourceTag/key-name`,

`aws:RequestTag/key-name` 또는 `aws:TagKeys` 조건 키를 사용하여 정책의 조건 요소에 태그 정보를 제공합니다. 이러한 조건 키에 대한 자세한 내용은 AWS 리소스 태그를 사용하여 리소스에 대한 액세스 제어를 참조하세요.

리소스의 태그를 기반으로 리소스에 대한 액세스를 제한하는 자격 증명 기반 정책의 예시는 태그에 기반한 X-Ray 그룹 및 샘플링 규칙에 대한 액세스 관리에서 확인할 수 있습니다.

문제 해결 AWS X-Ray

이 문서는 X-Ray API, 콘솔 또는 SDK를 사용할 때 발생할 수 있는 공통 오류 및 문제를 나열합니다. 여기에 나열되지 않은 문제를 발견하는 경우 이 페이지의 [Feedback] 버튼을 사용하여 해당 문제를 보고할 수 있습니다.

Sections

X-Ray 트레이스 맵 및 트레이스 세부 정보 페이지

다음 단원은 X-Ray 트레이스 맵 및 트레이스 세부 정보 페이지를 사용하는 데 문제가 있는 경우 도움이 될 수 있습니다.

CloudWatch 로그가 모두 표시되지 않음

X-Ray 트레이스 맵 및 트레이스 세부 정보 페이지에 표시되도록 로그를 구성하는 방법은 서비스에 따라 다릅니다.

- API Gateway에 로깅이 활성화되어 있다면 API Gateway 로그가 나타납니다.

모든 서비스 맵 노드에서 관련 로그 보기를 지원하는 것은 아닙니다. 다음 노드 유형에 대한 로그를 볼 수 있습니다.

- Lambda 컨텍스트
- Lambda 함수
- API 게이트웨이 스테이지
- Amazon ECS 클러스터
- Amazon ECS 인스턴스
- Amazon ECS 서비스
- Amazon ECS 태스크

- Amazon EKS 클러스터
- Amazon EKS 네임스페이스
- Amazon EKS 노드
- Amazon EKS 포드
- Amazon EKS 서비스

X-Ray 트레이스 맵에 모든 경보가 표시되지 않음

X-Ray 트레이스 맵에는 노드와 연관된 경보가 ALARM 상태인 경우에만 해당 노드의 경보 아이콘이 표시됩니다.

트레이스 맵은 다음 로직을 사용하여 경보를 노드와 연결합니다.

- 노드가 AWS 서비스를 나타내는 경우 해당 서비스와 연결된 네임스페이스가 있는 모든 경보가 노드와 연결됩니다. 예를 들어 `AWS::Kinesis` 유형의 노드는 CloudWatch 네임스페이스 `AWS/Kinesis`의 지표를 기반으로 하는 모든 경보와 연결됩니다.
- 노드가 AWS 리소스를 나타내는 경우 해당 특정 리소스에 대한 경보가 연결됩니다. 예를 들어 이름이 “MyTable”인 `AWS::DynamoDB::Table` 유형의 노드는 네임스페이스 `AWS/DynamoDB`가 있는 지표를 기반으로 하며 `TableName` 차원이 `MyTable`로 설정된 모든 경보에 연결됩니다.
- 노드가 이름 주위의 테두리가 파선으로 식별되는 알 수 없는 유형이면 해당 노드와 연결된 경보가 없습니다.

트레이스 맵에 일부 AWS 리소스가 표시되지 않음

모든 AWS 리소스가 전용 노드로 표시되는 것은 아닙니다. 일부 AWS 서비스는 서비스에 대한 모든 요청에 대해 단일 노드로 표시됩니다. 다음 리소스 유형은 리소스별 노드와 함께 표시됩니다.

- `AWS::DynamoDB::Table`
- `AWS::Lambda::Function`

  Lambda 함수는 두 개의 노드로 표시되는데, 하나는 Lambda 컨테이너용 노드이고 다른 하나는 함수용 노드입니다. 이렇게 하면 Lambda 함수의 콜드 스타트 문제를 식별하는 데 도움이 됩니다. Lambda 컨테이너 노드는 Lambda 함수 노드와 동일한 방식으로 경보 및 대시보드에 연결됩니다.
- `AWS::ApiGateway::Stage`
- `AWS::SQS::Queue`

- `AWS::SNS::Topic`

트레이스 맵에 노드가 너무 많음

X-Ray 그룹을 사용하여 맵을 여러 맵으로 분할합니다. 자세한 내용은 그룹에 필터 표현식 사용을 참조하십시오.

Java용 AWS X-Ray SDK

오류: 스레드 “Thread-1"에서 예외가 발생했습니다. amazonaws.xray.Exceptions.segmentNotFoundException: 'AmazonSNS'라는 이름의 하위 세그먼트를 시작하지 못했습니다. 세그먼트를 찾을 수 없습니다.

이 오류는 X-Ray SDK가에 대한 발신 호출을 기록하려고 시도 AWS했지만 열린 세그먼트를 찾을 수 없음을 나타냅니다. 이는 다음과 같은 상황에서 발생할 수 있습니다.

- servlet 필터가 구성되지 않음 - X-Ray SDK는 `AWSXRayServletFilter`라는 필터를 사용하여 수신 요청에 대한 세그먼트를 생성합니다. 수신 요청을 계측하도록 servlet 필터를 구성합니다.
- servlet 코드 외부에서 계측된 클라이언트를 사용하는 경우 — 계측된 클라이언트를 사용하여 시작 코드 또는 수신 요청에 대한 응답으로 실행되지 않는 기타 코드에서 직접 호출을 수행하는 경우 세그먼트를 수동으로 만들어야 합니다. 시작 코드 구성의 예제를 참조하세요.
- 작업자 스레드에서 계측된 클라이언트를 사용하는 경우 — 새 스레드를 생성하면 X-Ray 레코더가 열린 세그먼트에 대한 참조를 잃게 됩니다. `getTraceEntity` 및 `setTraceEntity` 메서드를 사용하여 현재 세그먼트 또는 하위 세그먼트 (`Entity`) 에 대한 참조를 가져와서 스레드 내부의 레코더로 다시 전달할 수 있습니다. 예제는 작업자 스레드에서 구성된 클라이언트 사용 섹션을 참조하세요.

Node.js용 X-Ray SDK

문제: CLS가 Sequelize에서 작동하지 않습니다.

`cls` 메서드를 사용하여 Node.js용 X-Ray SDK 네임스페이스를 Sequelize에 전달합니다.

```
var AWSXRay = require('aws-xray-sdk');
const Sequelize = require('sequelize');
Sequelize.cls = AWSXRay.getNamespace();
const sequelize = new Sequelize(...);
```

문제: CLS가 블루버드에서 작동하지 않습니다.

`cls-bluebird`을 사용하여 블루버드가 CLS와 함께 작동하도록 합니다.

```
var AWSXRay = require('aws-xray-sdk');
var Promise = require('bluebird');
var clsBluebird = require('cls-bluebird');
clsBluebird(AWSXRay.getNamespace());
```

X-Ray 대몬(daemon)

문제: 대몬(daemon)이 잘못된 보안 인증 정보를 사용하고 있습니다.

데몬은 AWS SDK를 사용하여 자격 증명을 로드합니다. 여러 가지 보안 인증 정보 제공 방법을 사용하는 경우 우선 순위가 가장 높은 방법이 사용됩니다. 자세한 내용은 데몬 실행 섹션을 참조하세요.

에 대한 문서 기록 AWS X-Ray

다음 표에서는 설명서의 중요 변경 사항을 설명합니다 AWS X-Ray. 이 설명서에 대한 업데이트 알림을 받으려면 RSS 피드를 구독하면 됩니다.

최신 설명서 업데이트: 2024년 3월 07일

변경 사항	설명	날짜
추가된 기능	X-Ray에서 OpenTelemetry로 마이그레이션. 자세한 내용은 X-Ray 계측에서 OpenTelemetry 계측으로 마이그레이션을 참조하세요.	2025년 6월 13일
추가된 기능	AWS X-Ray는 이제 트랜잭션 검색을 지원합니다. 자세한 내용은 트랜잭션 검색 섹션을 참조하세요.	2024년 11월 21일
추가된 기능	AWS X-Ray는 이제 OpenTelemetry Protocol(OTLP) 엔드포인트를 지원합니다. 자세한 내용은 OpenTelemetry를 참조하세요.	2024년 11월 21일
추가된 기능	X-Ray는 이제 , 및 `PutTraceSegments` `GetTraceSummaries` 를 포함한 데이터 이벤트를에 로깅`BatchGetTraces` 합니다 AWS CloudTrail. X-Ray는 또한 `GetSamplingStatisticSummaries` 관리 이벤트도 CloudTrail 에 로깅합니다. 자세한 내용은 를 사용하	2024년 3월 7일

	여 X-Ray API 호출 로깅 AWS CloudTrail을 참조하세요.	
추가된 기능	X-Ray는 이제 OpenTelemetry 또는 W3C 추적 컨텍스트 사양을 준수하는 다른 프레임워크를 통해 생성된 추적 ID를 지원합니다. 자세한 내용은 X-Ray로 추적 데이터 전송을 참조하세요.	2023년 10월 25일
추가된 기능	이제 Amazon SNS 활성 추적을 구성하여 요청이 Amazon SNS 토픽을 통과할 때 요청을 추적하고 분석할 수 있습니다. 자세한 내용은 Amazon SNS 및 AWS X-Ray 섹션을 참조하세요.	2023년 2월 8일
Node.js용 X-Ray SDK 주제 업데이트	AWS SDK for JavaScript V3를 사용하여 클라이언트를 계측하기 위한 세부 정보가 추가되었습니다. 자세한 내용은 Node.js용 X-Ray AWS SDK를 사용하여 SDK 호출 추적을 참조하세요.	2023년 2월 7일
IAM 관리형 정책 세부 정보 업데이트	`AWSXRayReadOnlyAccess` , `AWSXRayFullAccess` 및 `AWSXrayCrossAccountSharingConfiguration` 의 관리형 정책에 계정 간 관찰성을 위한 IAM 권한을 추가했습니다. 자세한 내용은 X-Ray의 IAM 관리형 정책을 참조하십시오.	2023년 2월 7일

추가된 기능	AWS X-Ray 는 이제 교차 계정 관찰성을 지원하므로 내의 여러 계정에 걸쳐 있는 애플리케이션을 모니터링하고 문제를 해결할 수 있습니다 AWS 리전. 자세한 내용은 교차 계정 추적을 참조하십시오.	2022년 11월 27일
추가된 기능	이제 메시지 생산자, Amazon SQS 대기열 및 소비자 간에 연결된 트레이스를 볼 수 있어 이벤트 기반 애플리케이션에서 전송된 트레이스를 연결된 뷰로 볼 수 있습니다. 자세한 내용은 이벤트 중심 애플리케이션 트레이싱을 참조하세요.	2022년 11월 20일
IAM 관리형 정책 세부 정보 업데이트	`AWSXRayReadOnlyAccess` 관리형 정책에 리소스 정책을 나열할 수 있는 IAM 권한을 추가했습니다. 자세한 내용은 X-Ray의 IAM 관리형 정책을 참조하십시오.	2022년 11월 15일
IAM 콘솔 권한 및 관리형 정책 세부 정보 업데이트	X-Ray 콘솔에서 사용하는 IAM 권한 집합이 `AWSXRayReadOnlyAccess` 관리형 정책에 대한 설명과 함께 업데이트되었습니다. 자세한 내용은 X-Ray 콘솔 사용을 참조하십시오.	2022년 11월 11일

AWS Distro for OpenTelemetry Ruby 추가	AWS Distro for OpenTelemetry(ADOT)는 분산 추적 및 지표를 수집하기 위한 단일 오픈 소스 APIs, 라이브러리 및 에이전트 세트를 제공합니다. ADOT Ruby를 사용하면 X-Ray 및 기타 추적 백엔드용 Ruby 애플리케이션을 계측할 수 있습니다. 자세한 내용은 AWS 배포판 OpenTelemetry Ruby를 참조하세요.	2022년 2월 7일
추가된 기능	이제 CloudWatch 콘솔에서 트레이스를 확인하고 X-Ray를 구성할 수 있습니다. 자세한 내용은 X-Ray 콘솔 섹션을 참조하십시오.	2022년 1월 24일
CloudWatch RUM 통합	AWS X-Ray 및 CloudWatch RUM을 사용하면 애플리케이션의 최종 사용자부터 다운스트림 AWS 관리형 서비스를 통해 요청 경로를 분석하고 디버깅할 수 있습니다. 자세한 내용은 CloudWatch RUM 및 AWS X-Ray 섹션을 참조하세요.	2021년 12월 3일
OpenTelemetry용 통합 AWS 배포판	AWS Distro for OpenTelemetry(ADOT)는 분산 추적 및 지표를 수집하기 위한 단일 오픈 소스 APIs, 라이브러리 및 에이전트 세트를 제공합니다. ADOT를 사용하면 X-Ray 및 기타 추적 백엔드용 애플리케이션을 계측할 수 있습니다. 자세한 내용은 애플리케이션 계측하기를 참조하세요.	2021년 9월 23일

추가된 기능	AWS X-Ray 이제가 Amazon Virtual Private Cloud와 통합되어 Amazon VPC의 리소스가 퍼블릭 인터넷을 통하지 않고 X-Ray 서비스와 통신할 수 있습니다. 자세한 내용은 VPC 엔드포인트 AWS X-Ray 와 함께 사용을 참조하세요.	2021년 5월 20일
추가된 기능	AWS X-Ray 이제가와 통합되어 X-Ray 리소스를 프로비저닝하고 구성할 수 AWS CloudFormation있습니다. 자세한 내용은 CloudFormation으로 X-Ray 리소스 생성을 참조하세요.	2021년 5월 6일
추가된 기능	AWS X-Ray 이제가 Amazon EventBridge와 통합되어 EventBridge를 통해 전달되는 이벤트를 추적합니다. 이를 통해 사용자는 시스템을 보다 완벽하게 파악할 수 있습니다. 자세한 내용은 Amazon EventBridge 및 AWS X-Ray 섹션을 참조하세요.	2021년 3월 2일
ECR에 대몬(daemon)을 추가했습니다.	이제 Amazon ECR에서 대몬(daemon)을 다운로드할 수 있습니다. 자세한 내용은 대몬(daemon) 다운로드하기를 참조하세요.	2021년 3월 1일

추가된 기능	AWS X-Ray 는 이제 Amazon EventBridge에 대한 인사이트 관련 알림을 지원합니다. 이를 통해 EventBridge를 사용하여 인사이트에 대한 자동 조치를 취할 수 있습니다. 자세한 내용은 인사이트 알림을 참조하십시오.	2020년 10월 15일
다운로드 가능한 대몬(daemon)이 추가되었습니다.	AWS X-Ray 에는 Linux ARM64용 지원 데몬이 도입되었습니다. 자세한 내용은 AWS X-Ray daemonbrazil ws 를 참조하십시오.	2020년 10월 1일
추가된 기능	AWS X-Ray 는 이제 Amazon CloudWatch Synthetics와의 활성 통합을 지원합니다. 이를 통해 응답 시간 및 상태와 같은 Synthetics canary 클라이언트 노드에 대한 세부 정보를 확인할 수 있습니다. 또한 Synthetics canary 클라이언트 노드의 정보를 기반으로 분석 콘솔에서 분석을 수행할 수 있습니다. 자세한 내용은 X-Ray를 사용하여 CloudWatch Synthetics Canaries 디버깅하기를 참조하십시오.	2020년 9월 24일

추가된 기능	AWS X-Ray 는 이제 end-to-end 워크플로 추적을 지원합니다 AWS Step Functions. 상태 시스템 구성 요소를 시각화하고 성능 병목 현상을 식별하며 오류가 발생한 요청 문제를 해결할 수 있습니다. 자세한 내용은 AWS Step Functions 및 AWS X-Ray 단원을 참조하십시오.	2020년 9월 14일
추가된 기능	AWS X-Ray 는 계정의 추적 데이터를 지속적으로 분석하여 애플리케이션의 긴급한 문제를 식별하는 인사이트를 제공합니다. Insights는 사고를 기록하고 해결될 때까지 사고 영향을 추적합니다. 자세한 내용은 AWS X-Ray 콘솔에서 인사이트 사용을 참조하세요.	2020년 9월 3일
추가된 기능	AWS X-Ray 는 Java 자동 계측 에이전트를 도입하여 고객이 기존 Java 기반 애플리케이션을 수정하지 않고도 트레이스 데이터를 수집할 수 있도록 합니다. 이제 구성 변경을 최소화하고 코드를 변경하지 않고도 Java 웹 및 서브렛 기반 응용 프로그램을 추적할 수 있습니다. 자세한 내용은 Java용AWS X-Ray 자동 계측 에이전트를 참조하십시오.	2020년 9월 3일

추가된 기능	AWS X-Ray 는 추적 그룹을 쉽게 생성하고 관리할 수 있도록 X-Ray 콘솔에 새 그룹 페이지를 추가했습니다. 자세한 내용은 X-Ray 콘솔에서 그룹 구성하기을 참조하십시오.	2020년 8월 24일
추가된 기능	AWS X-Ray 이제에서 그룹 및 샘플링 규칙에 태그를 추가할 수 있습니다. 또한 태그를 기반으로 그룹 및 샘플링 규칙에 대한 액세스를 제어할 수 있습니다. 자세한 내용은 X-Ray 샘플링 규칙 및 그룹에 태그 지정 및 태그에 따른 X-Ray 그룹 및 샘플링 규칙에 대한 액세스 관리를 참조하십시오.	2020년 8월 24일

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.