



개발자 안내서

# AWS X-Ray



# AWS X-Ray: 개발자 안내서

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

Amazon의 상표 및 브랜드 디자인은 Amazon 외 제품 또는 서비스와 함께, 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

무엇입니까 AWS X-Ray? .....	1
X-Ray의 작동 방식 .....	1
X-Ray가 계속 애플리케이션과 상호 작용하는 방식 .....	2
개념 .....	6
세그먼트 .....	6
하위 세그먼트 .....	7
서비스 그래프 .....	10
트레이스 .....	12
샘플링 .....	13
추적 헤더 .....	14
필터 표현식 .....	15
그룹 .....	15
주석 및 메타데이터 .....	16
오류, 결함 및 예외 .....	16
시작 .....	18
인터페이스 선택 .....	20
사용 AWS Management Console .....	21
Amazon CloudWatch 콘솔 사용 .....	22
X-Ray 콘솔 사용하기 .....	22
X-Ray 콘솔 살펴보기 .....	23
SDK 사용 .....	91
ADOTSDK 사용 .....	92
SDK 사용 X-Ray .....	93
X-Ray API 사용 .....	94
X-Ray API .....	96
X-Ray 대몬(daemon) .....	150
데몬 다운로드 .....	150
데몬 아카이브의 서명 확인 .....	152
데몬 실행 .....	153
대몬(daemon)에 X-Ray로 데이터를 전송할 권한 부여 .....	153
X-Ray 대몬(daemon) 로그 .....	154
구성 .....	155
지원되는 환경 변수 .....	155
명령줄 옵션 사용 .....	156

구성 파일 사용 .....	157
로컬에서 데몬 실행 .....	158
Linux에서 X-Ray 데몬(daemon) 실행 .....	159
(Docker 컨테이너에서 X-Ray 데몬(daemon) 실행 .....	159
Windows에서 X-Ray 데몬(daemon) 실행 .....	160
OS X에서 X-Ray 데몬(daemon) 실행 .....	161
Elastic Beanstalk에서 .....	162
Elastic Beanstalk X-Ray 통합을 사용하여 X-Ray 데몬(daemon) 실행하기 .....	162
수동으로 X-Ray 데몬(daemon) 다운로드 및 실행하기 (고급) .....	164
Amazon EC2에서 .....	166
Amazon ECS에서 .....	167
공식 도커 이미지 사용 .....	167
도커 이미지 생성 및 빌드 .....	168
Amazon ECS 콘솔에서 명령줄 옵션 구성 .....	171
애플리케이션을 계측하세요. ....	173
배포판을 사용하여 애플리케이션을 계측하기 AWS OpenTelemetry .....	173
AWS X-Ray SDK로 애플리케이션 계측하기 .....	175
AWS 배포판용 OpenTelemetry SDK와 X-Ray SDK 중 선택 .....	176
다음과 같은 기기 Go .....	176
AWS Distro for OpenTelemetry Go .....	177
Go용 X-Ray SDK .....	177
다음과 같은 기기 Java .....	193
AWS Distro for OpenTelemetry Java .....	193
엑스레이 SDK를 위한 Java .....	194
다음과 같은 기기 Node.js .....	246
AWS Distro for OpenTelemetry JavaScript .....	247
Node.js용 X-Ray SDK .....	247
다음과 같은 기기 Python .....	271
AWS 배포판 대상 OpenTelemetry Python .....	272
엑스레이 SDK를 위한 Python .....	272
다음과 같은 기기 .NET .....	303
AWS 배포판 대상 OpenTelemetry .NET .....	303
.NET용 X-Ray SDK .....	303
Ruby를 사용한 인스트루먼트 .....	328
AWS Distro for OpenTelemetry Ruby .....	329
Ruby용 X-Ray SDK .....	329

다음과 통합 AWS 서비스 .....	347
AWS Distro for OpenTelemetry .....	349
AWS Distro for OpenTelemetry .....	349
API Gateway .....	350
App Mesh .....	352
App Runner .....	355
AWS AppSync .....	355
CloudTrail .....	355
의 X-Ray 관리 이벤트 CloudTrail .....	357
의 X-Ray 데이터 이벤트 CloudTrail .....	357
X-Ray 이벤트 예제 .....	359
CloudWatch .....	361
CloudWatch 램 .....	362
CloudWatch Synthetics .....	363
AWS Config .....	372
Lambda 함수 트리거 생성 .....	373
X-Ray용 사용자 지정 AWS Config 규칙 생성 .....	374
결과 예 .....	375
Amazon SNS 알림 .....	375
Amazon EC2 .....	375
Elastic Beanstalk .....	376
Elastic Load Balancing .....	376
EventBridge .....	377
X-Ray 서비스 맵에서 소스 및 대상 보기 .....	377
추적 컨텍스트를 이벤트 대상으로 전파하기 .....	377
Lambda .....	384
Amazon SNS .....	385
Amazon SNS 활성 추적 구성하기 .....	386
X-Ray 콘솔에서 Amazon SNS 게시자 및 구독자 추적 보기 .....	387
Step Functions .....	389
Amazon SQS .....	390
HTTP 추적 헤더 전송 .....	391
추적 헤더 검색 및 추적 컨텍스트 복구 .....	392
Amazon S3 .....	393
Amazon S3 이벤트 알림 구성 .....	393
리소스 관리 .....	395

를 사용하여 X-Ray 리소스 만들기 CloudFormation .....	396
X-Ray 및 AWS CloudFormation 템플릿 .....	396
에 대해 자세히 알아보기 AWS CloudFormation .....	396
태그 지정 .....	396
태그 제한 .....	398
콘솔에서 태그 관리 .....	398
내 태그 관리 AWS CLI .....	400
태그를 기반으로 X-Ray 리소스에 대한 액세스 제어 .....	404
샘플 애플리케이션 .....	405
Scorekeep 튜토리얼 .....	407
필수 조건 .....	408
다음을 사용하여 Scorekeep 애플리케이션을 설치합니다. CloudFormation .....	409
데이터 추적 생성 .....	410
에서 트레이스 맵을 확인하세요. AWS Management Console .....	411
Amazon SNS 알림 구성 .....	419
샘플 애플리케이션 탐색 .....	420
선택 사항: 최소 권한 정책 .....	425
정리 .....	428
다음 단계 .....	428
AWS SDK 클라이언트 .....	429
사용자 지정 하위 세그먼트 .....	430
주석 및 메타데이터 .....	430
HTTP 클라이언트 .....	431
SQL 클라이언트 .....	432
AWS Lambda 함수 .....	434
Random name .....	435
작업자 .....	437
시작 코드 구성 .....	439
스크립트 구성 .....	441
웹 클라이언트 구성하기 .....	443
작업자 스레드 .....	447
문제 해결 .....	449
X-Ray 트레이스 맵 및 트레이스 세부 정보 페이지 .....	449
모든 CloudWatch 로그가 보이지 않아요 .....	449
X-Ray 추적 맵에 모든 알람이 표시되지 않습니다. ....	450
트레이스 맵에 일부 AWS 리소스가 보이지 않습니다. ....	450

추적 맵에 노드가 너무 많습니다. ....	451
Java용 X-Ray SDK .....	451
Node.js용 X-Ray SDK .....	451
X-Ray 대몬(daemon) .....	452
보안 .....	453
.....	453
데이터 보호 .....	453
자격 증명 및 액세스 관리 .....	456
고객 .....	456
자격 증명을 통한 인증 .....	457
정책을 사용한 액세스 관리 .....	459
AWS X-Ray IAM과의 작동 방식 .....	461
자격 증명 기반 정책 예시 .....	469
문제 해결 .....	482
로깅 및 모니터링 .....	484
규정 준수 확인 .....	484
복원성 .....	485
인프라 보안 .....	486
VPC 엔드포인트 .....	486
X-Ray용 VPC 엔드포인트 생성 .....	487
X-Ray VPC 엔드포인트에 대한 액세스 제어 .....	488
지원되는 리전 .....	489
문서 기록 .....	491
.....	cdxcviii

# 무엇입니까 AWS X-Ray?

AWS X-Ray 다음을 포함하여 인스트루먼트된 애플리케이션이 보내는 모든 수신 응답 및 호출에 대한 추적 정보를 제공합니다.

- 다운스트림 리소스 AWS
- 마이크로서비스
- 데이터베이스 수
- 웹 API

추적 데이터 및 시각화를 사용하여 애플리케이션 성능에 대한 통찰력을 얻고, 문제를 식별하고, 최적화 기회를 찾을 수 있습니다. X-Ray의 분석 도구를 사용하여 애플리케이션에 대한 추적된 요청의 세부 정보를 보고, 필터링하고, 조사할 수 있습니다.

## X-Ray의 작동 방식

X-Ray를 사용하려면 먼저 애플리케이션을 계측하여 X-Ray가 애플리케이션이 요청을 처리하는 방식을 추적할 수 있도록 해야 합니다. 애플리케이션에 계측을 추가하면 X-Ray가 애플리케이션 내의 수신 및 아웃바운드 요청과 기타 이벤트에 대한 추적 데이터와 메타데이터를 전송할 수 있습니다. 예를 들어 Java 애플리케이션이 보내는 모든 수신 HTTP 요청과 다운스트림 호출을 AWS 서비스 계측할 수 있습니다. 애플리케이션을 자동으로 계측할 수도 있습니다. 자세한 내용은 [애플리케이션 계측](#)을 참조하십시오.

X-Ray는 계측 애플리케이션이 수신하는 모든 요청에 추적 ID를 할당합니다. 응용 프로그램이 다른 구성 요소와 상호 작용하는 경우 X-Ray는 세그먼트를 만듭니다. 이 세그먼트는 원래 추적 ID와 연결되며 해당 구성 요소와의 상호 작용 품질을 추적합니다.

X-Ray는 전체 애플리케이션 워크플로우에서 추적 ID 및 세그먼트를 추적합니다. 전체 워크플로를 분석하거나 세부 분석을 위해 일부를 분리할 수 있습니다. 세그먼트에 대한 자세한 내용은 다음 [개념](#) 섹션을 참조하십시오.

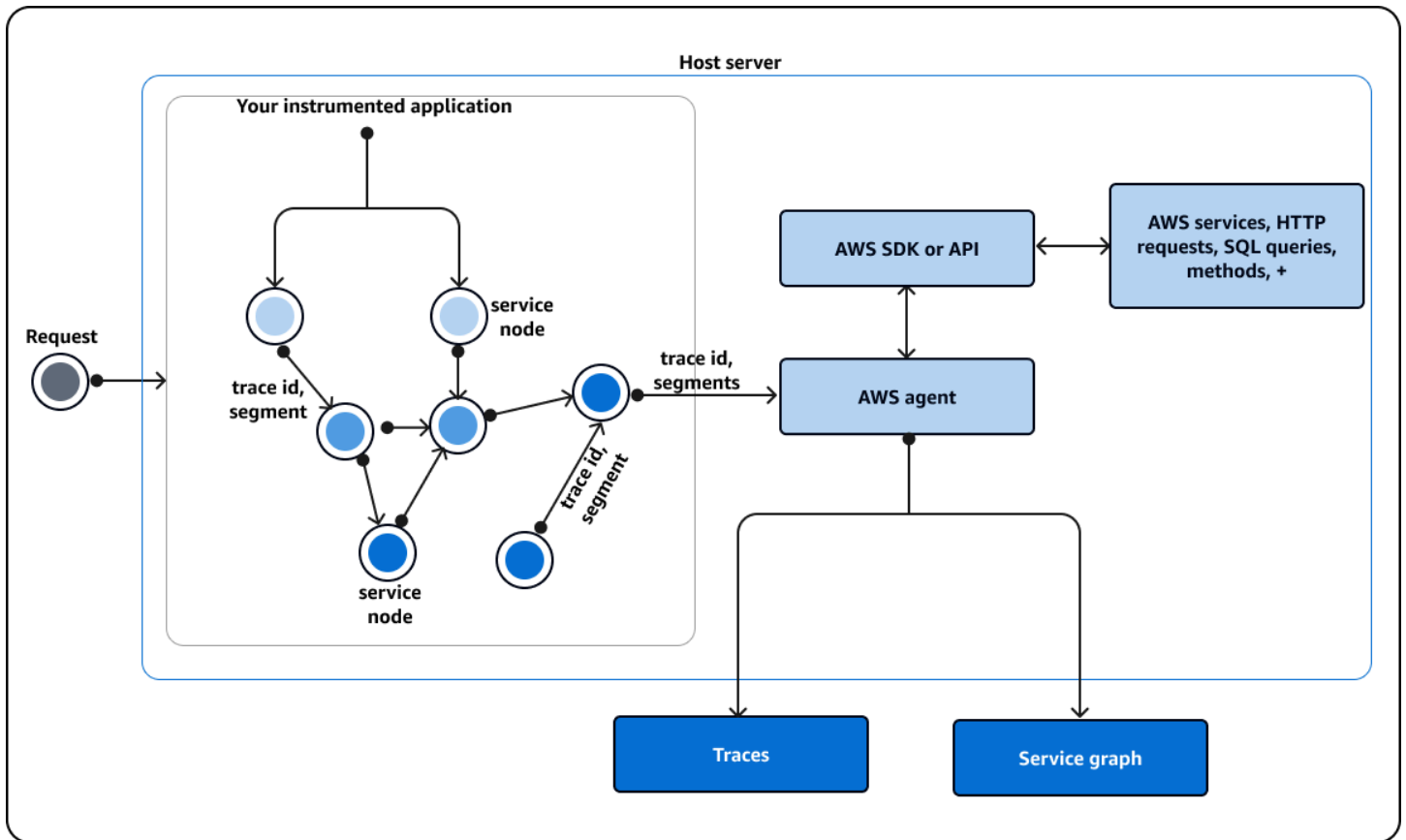
X-Ray는 서비스 노드 또는 구성 요소와 상호 작용하는 애플리케이션을 추적하여 다음과 같이 들어오는 요청을 처리합니다.

1. X-Ray는 트레이스 ID와 세그먼트를 사용하여 개별 상호 작용을 추적합니다.
2. AWS 에이전트는 추적 ID 및 관련 세그먼트를 수집한 다음 이를 SDK 또는 API 추적 프레임워크로 전달합니다.



- 또한 X-Ray는 X-Ray와 통합되는 모든 AWS 서비스와의 상호 작용을 추적합니다.
- 에이전트는 콘솔 GUI로 데이터를 전송합니다. 콘솔 GUI에서 트레이스, 세그먼트 및 하위 세그먼트에 대한 정보와 이러한 구성 요소가 상호 작용하는 방식을 볼 수 있습니다.

이전 단계는 다음 다이어그램에 나와 있습니다.



## X-Ray가 계측 애플리케이션과 상호 작용하는 방식

계측 애플리케이션이 요청을 받으면 X-Ray는 다음을 수행합니다.

- 애플리케이션이 요청을 처리한 후, X-Ray SDK는 추적 데이터를 AWS 수집기 또는 에이전트에 보냅니다. 그런 다음 에이전트는 추적 ID와 세그먼트를 수집합니다. 다음 세 에이전트 중에서 선택할 수 있습니다.
  - [AWS 배포판 \(Distro for\) OpenTelemetry \(ADOT\) Collector](#) — 표준화된 오픈 소스 에이전트를 기반으로 최적화되고 보호되는 AWS 오픈 소스 컬렉터입니다. [OpenTelemetry](#) 언어와 벤더에 구애받지 않는 표준화된 코드를 사용하여 에이전트와 상호 작용하면서도 최종 제품에 내장된 보안 및 최적화에 대한 확신을 갖고 ADOT Collector 싶다면 를 사용하십시오. AWS 를 사용하여 다양한 에이전트와 백엔드에 ADOT 대한 엔드포인트를 구성할 수도 있습니다.

- [Amazon CloudWatch agent](#) — 로그, 지표 및 추적을 통합하고 모든 원격 측정 데이터를 지원하며 여기에 ADOT Collector 통합된 오픈 소스 수집기입니다.
  - [X-Ray 데몬](#) — X-Ray SDK 및 X-Ray API와 함께 작동하는 컬렉터입니다. 레거시 코드가 있거나 사용자 지정 추적이 필요하여 X-Ray API를 사용해야 하는 응용 프로그램이 있는 경우 X-Ray 데몬을 사용하십시오. 데몬은 Linux Microsoft Windows macOS, 및 플랫폼에서 사용할 수 있으며 모든 플랫폼에 포함됩니다. AWS Elastic Beanstalk AWS Lambda
2. 그런 다음 에이전트는 이 데이터를 AWS API 또는 API를 기반으로 구축된 AWS SDK로 구성된 추적 프레임워크로 보냅니다. [이 프레임워크는 다른 서비스와 상호 작용합니다.](#) [AWS X-Ray API](#) 는 AWS SDK를 통해 또는 직접 모든 X-Ray 기능에 액세스할 수 있도록 합니다. AWS Command Line Interface HTTPS 언어를 사용하거나 SDK에서 지원하지 않는 작업이 필요한 경우 X-Ray API 를 사용하세요.

다음 SDK를 사용할 수 있습니다.

- ADOT SDK — ADOT SDK를 사용하여 제휴하지 않은 공급업체의 다른 에이전트와 상호 작용하세요. AWS 또한 ADOT SDK는 여러 백엔드 서비스를 지원합니다.
- X-Ray SDK — 더 이상 기능이나 언어를 추가하지 않는 클래식 제품입니다. 애플리케이션 코드를 업데이트하지 않으려면 X-Ray SDK를 사용하십시오.

X-Ray 또는 ADOT SDK를 사용하는 경우 에이전트와 함께 사용할 수 있는 옵션은 다음과 같습니다.

- CloudWatch 에이전트와 함께 X-Ray 또는 ADOT SDK를 사용하는 것이 좋습니다.
  - 보안 및 최적화 AWS 계층을 갖춘 벤더에 구매받지 않는 소프트웨어를 사용하려면 ADOT SDK 를 ADOT Collector —와 함께 사용하는 것이 좋습니다.
  - 에이전트와 함께 X-Ray SDK 사용 — CloudWatch CloudWatch 에이전트는 X-Ray SDK와 호환됩니다.
  - X-Ray SDK를 X-Ray 데몬과 함께 사용 — X-Ray SDK를 계속 사용하려면 이 옵션을 사용하십시오.
3. (선택 사항) 추적 프레임워크는 다른 AWS 서비스, HTTP 서버, 기타 메서드 및 쿼리와 상호 작용할 수 있습니다. X-Ray와 통합되는 일부 AWS 서비스로는 Amazon EC2 Amazon SNS 및 API Gateway가 있습니다. SDK 또는 API는 이러한 상호 작용 중에 추적 데이터를 추적합니다.

[AWS X-Ray와 통합되는](#) 서비스는 들어오는 요청에 추적 헤더를 추가하거나, 추적 데이터를 X-Ray로 보내거나, 에이전트를 실행하여 추적 데이터를 수집할 수 있습니다. 예를 들어, 요청에 대한 추적 데이터를 Lambda 함수로 전송할 AWS Lambda 수 있습니다.

X-Ray와 호환되는 다른 서비스에 대한 자세한 내용은 [을 참조하십시오](#) [다른 AWS X-Ray 사람과 통합 AWS 서비스](#).

4. 콘솔에서 그래픽 사용자 인터페이스 (GUI) 에서 트레이스, 세그먼트 및 서브세그먼트에 대한 데이터를 볼 수 있습니다. 다음 옵션을 사용할 수 있습니다.
  - <https://console.aws.amazon.com/cloudwatch/> — 트레이스, 로그, 메트릭을 한 곳에서 볼 수 있는 GUI 환경입니다. X-Ray 서비스 맵과 레거시 CloudWatch ServiceLens 맵은 CloudWatch 콘솔 내의 X-Ray 추적 맵에 결합됩니다.
  - <https://console.aws.amazon.com/xray/home> — 트레이스에 대한 정보를 볼 수 있는 GUI 환경입니다. 트레이스에 대한 인사이트, 트레이스 맵, 서비스 맵, 분석을 포함하는 정보를 볼 수 있습니다. AWS에서는 더 이상 이 콘솔 환경을 개발하지 않습니다.

X-Ray는 애플리케이션이 상호 작용하는 AWS 리소스의 추적 데이터를 사용하여 상세한 추적 맵을 생성합니다. 추적 맵은 프런트 엔드 서비스가 단일 요청으로 호출하는 클라이언트, 프런트 엔드 서비스 및 백엔드 서비스를 보여줍니다. 추적 맵을 사용하여 ID 병목 현상, 지연 시간 급증 및 기타 문제를 해결하여 애플리케이션의 성능을 해결하거나 개선할 수 있습니다.

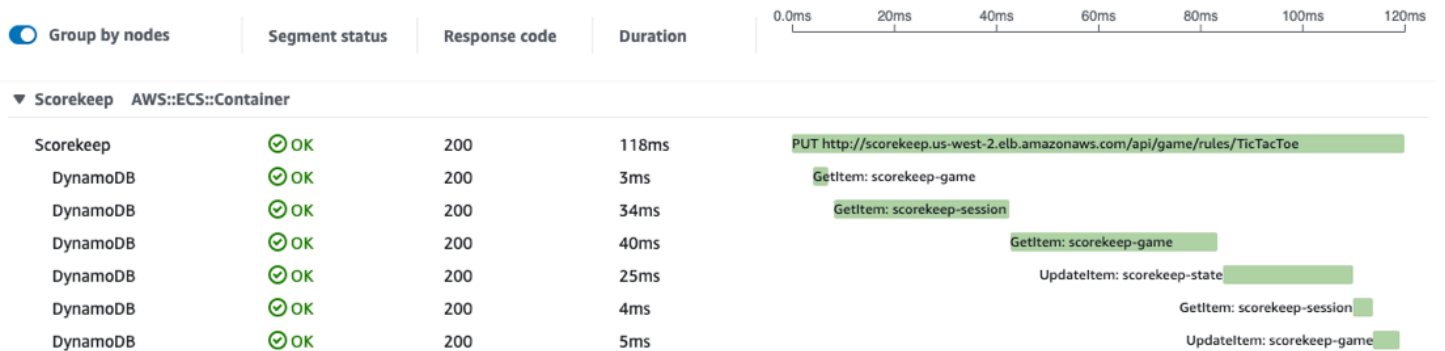
또한 X-Ray는 애플리케이션이 서비스 노드와 상호 작용하는 방식을 전체적으로 볼 수 있는 서비스 맵을 생성합니다. 서비스 맵의 에지는 서비스 노드를 연결합니다. 노드가 서로 통신하는 빈도와 해당 통신의 대기 시간을 보여줍니다.

다음 이미지는 애플리케이션이 다양한 구성 요소와 상호 작용하는 방식을 보여주는 서비스 맵의 예를 보여줍니다. 콘솔에서 서비스 맵을 볼 수 있습니다. 이 이미지는 클라이언트로부터 요청을 받는 애플리케이션을 보여줍니다. 그런 다음 이 이미지는 애플리케이션이 두 DynamoDB 테이블 및 Amazon SNS와 상호 작용하는 방식을 보여줍니다.



다음 이미지는 추적의 단일 세그먼트에 대해 콘솔에서 사용할 수 있는 데이터의 예입니다. 이 이미지는 여러 세그먼트를 나열한 타임라인과 각 세그먼트를 다른 세그먼트와 비교하여 실행한 시작 시간 및 기간을 보여줍니다. 이 이미지는 세그먼트 상태와 HTTP 응답 코드도 보여줍니다.

Segments Timeline [Info](#)



# 개념

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

## 개념

- [세그먼트](#)
- [하위 세그먼트](#)
- [서비스 그래프](#)
- [트레이스](#)
- [샘플링](#)
- [추적 헤더](#)
- [필터 표현식](#)
- [그룹](#)
- [주석 및 메타데이터](#)
- [오류, 결함 및 예외](#)

## 세그먼트

애플리케이션 로직을 실행하는 컴퓨팅 리소스는 작업에 대한 데이터를 [세그먼트](#)의 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, 내용.
- 완료된 작업 — 시작 및 종료 시간, 하위 세그먼트.
- 발생하는 문제 - 예외 스택의 자동 캡처를 포함한 [오류, 결함 및 예외](#)

다음 이미지는 세그먼트에 대해 반환된 개요 정보의 예시입니다. 이 이미지는 ID, 시작 및 종료 시간, 오류 또는 장애, HTTP 요청의 요청 및 응답 코드에 대한 정보를 보여줍니다.

## Segment details: Scorekeep



Overview	Resources	Annotations	Metadata	Exceptions	SQL
<b>Overview</b> Subsegment ID 1-12345678-5120cbe96265dfa965cba1ac-556f7a611a12900FF Name Scorekeep Origin AWS::ECS::Container			<b>Time</b> Start Time 2023-06-23 20:34:58.099 (UTC) End Time 2023-06-23 20:34:58.110 (UTC) Duration 11ms	<b>Errors and faults</b> Error false Fault false	<b>Requests &amp; Response</b> Request url http://scorekeep.us-west-2.elb.amazonaws.com/api/game/ Request method GET Response code 200

SDK 또는 API로 구성된 추적 프레임워크는 요청 및 응답 헤더, 애플리케이션의 코드, 애플리케이션이 실행되는 AWS 리소스에 대한 메타데이터에서 정보를 수집합니다. 들어오는 요청, 다운스트림 요청 및 AWS 서비스를 계측하도록 애플리케이션 구성 또는 코드를 수정하여 X-Ray에서 수집하는 데이터를 선택합니다.

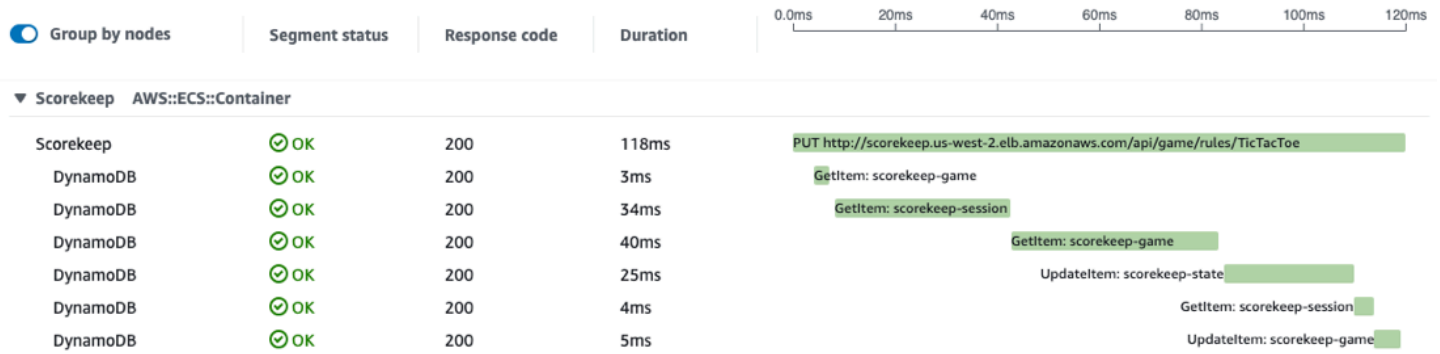
### 📘 전달된 요청

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 클라이언트 IP는 위조될 수 있으므로 신뢰하면 안 됩니다.

[SDK 또는 API와 같은 추적 프레임워크를 사용하여 주석 및 메타데이터를 포함한 추가 정보를 기록할 수 있습니다.](#) 세그먼트 및 하위 세그먼트 구조와 기록된 정보에 대한 자세한 내용은 [참조하십시오.](#) [X-Ray 세그먼트 문서](#) 세그먼트 문서의 크기는 최대 64kB까지 가능합니다.

## 하위 세그먼트

세그먼트를 하위 세그먼트로 나눌 수 있습니다. 하위 세그먼트는 애플리케이션이 원래 요청을 처리하기 위해 수행하는 다운스트림 호출에 대한 보다 세분화된 타이밍 정보와 세부 정보를 제공합니다. 하위 세그먼트에는 외부 HTTP API 또는 SQL 데이터베이스 호출에 대한 추가 세부 정보가 포함됩니다. AWS 서비스또한 서브세그먼트를 정의하여 애플리케이션의 특정 함수 또는 코드 라인을 계측할 수 있습니다.

Segments Timeline [Info](#)

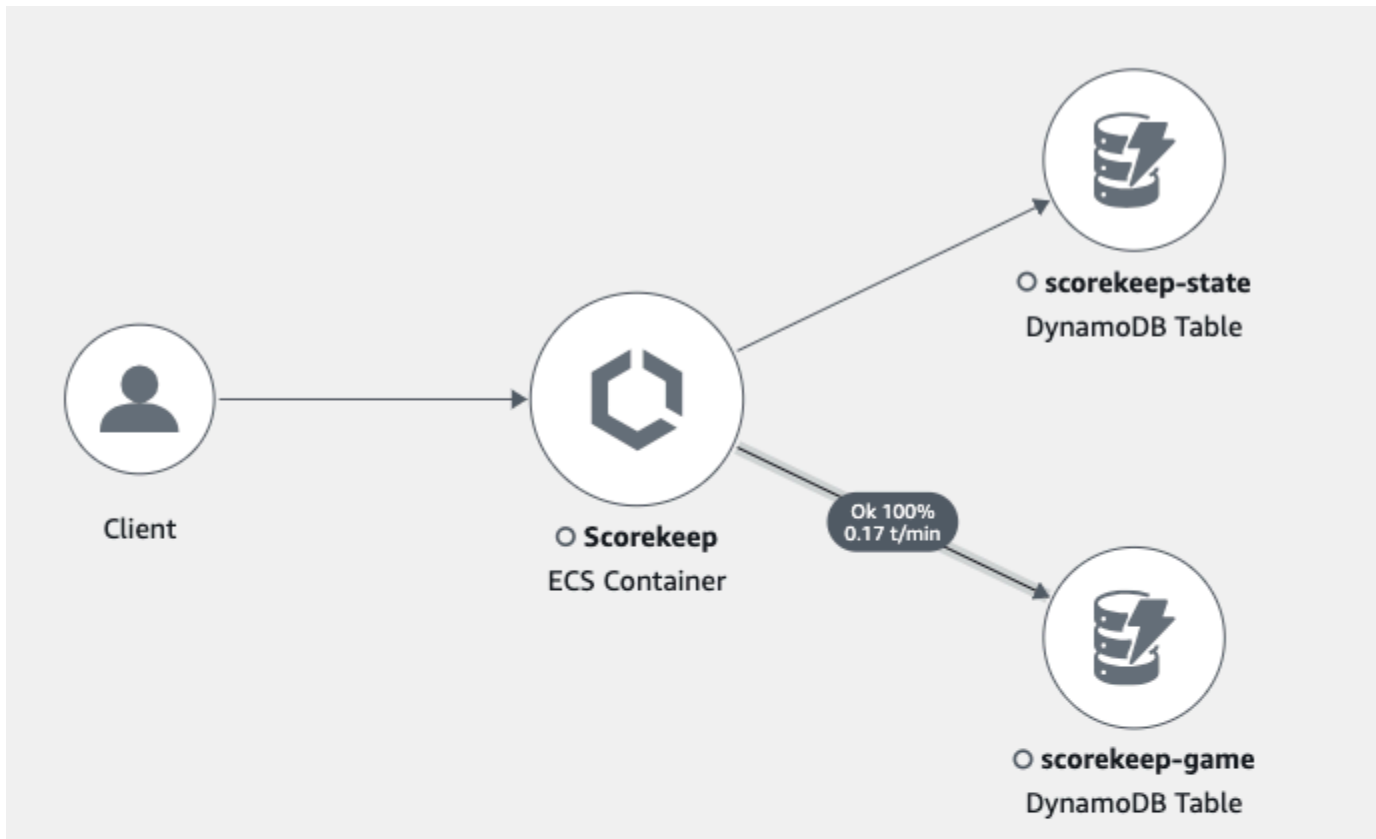
X-Ray는 Amazon DynamoDB와 같이 자체 세그먼트를 전송하지 않는 서비스의 추적 맵에서 하위 세그먼트를 사용하여 추론된 세그먼트와 다운스트림 노드를 생성합니다. 하위 세그먼트를 사용하면 종속성이 추적을 지원하지 않거나 외부에 있더라도 모든 다운스트림 종속성을 확인할 수 있습니다. AWS

하위 세그먼트는 다운스트림 호출의 애플리케이션 보기를 클라이언트로 표현합니다. 다운스트림 서비스도 인스트루먼트된 경우 해당 세그먼트는 업스트림 클라이언트의 하위 세그먼트에서 추론된 세그먼트를 대체합니다. 서비스 그래프의 노드는 가능한 경우 서비스 세그먼트의 정보를 사용합니다. 두 노드 사이의 엣지는 업스트림 서비스의 하위 세그먼트를 사용합니다.

예를 들어 AWS 계측 SDK 클라이언트를 사용하여 DynamoDB를 호출하면 X-Ray SDK는 해당 호출에 대한 하위 세그먼트를 기록합니다. DynamoDB는 세그먼트를 전송하지 않으므로 하위 세그먼트에는 다음에 대한 정보가 포함됩니다.

- 트레이스의 추론된 세그먼트.
- 서비스 그래프의 DynamoDB; 노드.
- 서비스와 DynamoDB 사이의 엣지.

다음 다이어그램은 샘플 애플리케이션의 서비스 맵을 보여줍니다. 이미지에서 클라이언트는 샘플 Scorekeep 애플리케이션을 요청합니다. 스코어킵 애플리케이션은 DynamoDB를 두 번 호출합니다. 서비스 맵의 엣지는 이러한 각 호출을 나타냅니다. Edge를 선택하면 DynamoDB 테이블에 대한 호출 상태, 횟수, 빈도를 확인할 수 있습니다. 다음 이미지는 응답 시간을 기준으로 필터링된 엣지에 해당하는 트레이스를 보여줍니다.



▼ Edge details

Source: Scorekeep Destination: scorekeep-game

Response time distribution filter

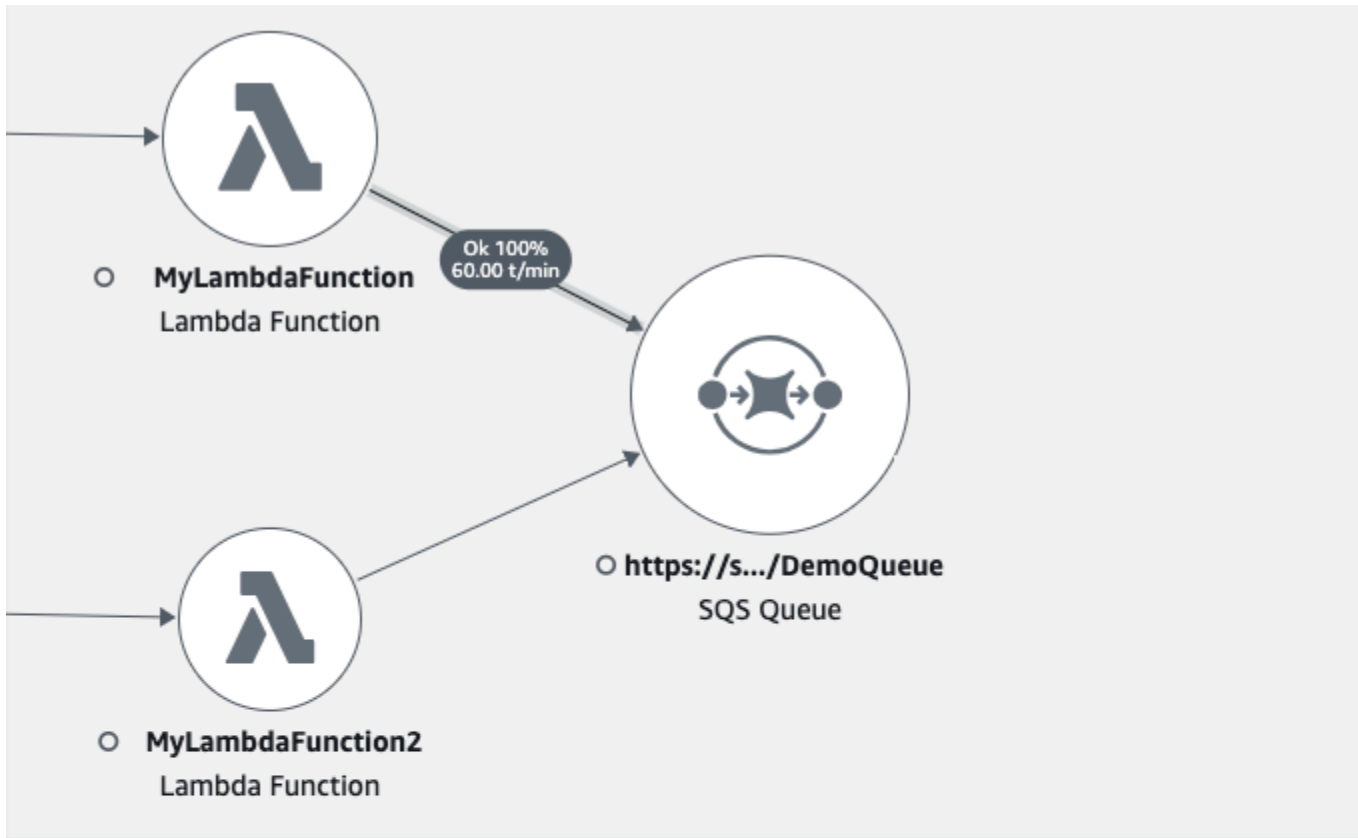
To filter traces by response time, select the corresponding area of the chart.



계측 애플리케이션으로 다른 계측 서비스를 호출하면 다운스트림 서비스가 자체 세그먼트를 전송합니다. 이 세그먼트는 업스트림 서비스가 하위 세그먼트에 기록한 것과 동일한 통화의 뷰를 기록합니다. 서비스 그래프에서 두 서비스의 노드 모두 해당 세그먼트의 타이밍 및 오류 정보를 포함합니다. 이들 사이의 경계에는 업스트림 서비스 하위 세그먼트의 정보가 포함됩니다. 다운스트림 서비스는 요청 작업을 시작하고 종료한 시간을 기록합니다. 업스트림 서비스는 요청이 두 서비스 사이를 이동하는 데 소요된 시간을 포함하여 왕복 지연 시간을 기록합니다.



다음 이미지는 업스트림 Lambda 함수에 해당하는 엣지에서 응답 시간을 기준으로 필터링된 추적 정보를 보여줍니다.



▼ Edge details

Source: MyLambdaFunction Destination: https://sqs.us-west-2.amazonaws.com/MySQSQueue

Response time distribution filter

To filter traces by response time, select the corresponding area of the chart.



서비스 그래프

X-Ray는 애플리케이션이 전송하는 데이터를 사용하여 서비스 그래프를 생성합니다. X-Ray로 데이터를 보내는 각 AWS 리소스는 그래프에서 서비스 노드로 나타납니다. 에지는 함께 작동하는 서비스를

연결하여 요청을 처리하고, 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.

다음 이미지는 서비스 맵을 보여줍니다. 서비스 맵은 애플리케이션에 대한 클라이언트의 요청과 애플리케이션이 요청을 처리하기 위해 상호 작용하는 서비스 간의 관계를 표시합니다. 다음 이미지에서 샘플 Scorekeep 애플리케이션은 두 개의 DynamoDB 테이블 및 Amazon SNS와 상호 작용합니다.



분산 애플리케이션에서 X-Ray는 동일한 추적 ID로 요청을 처리하는 모든 서비스의 노드를 단일 서비스 그래프로 결합합니다. 요청이 상호 작용하는 첫 번째 서비스는 프런트 엔드와 해당 서비스가 호출하는 서비스 간에 전파되는 [추적 헤더](#)를 추가합니다.

예를 들어 [Scorekeep](#)은 AWS Lambda 함수를 호출하여 임의의 이름을 생성하는 웹 API를 실행합니다. 그런 다음, X-Ray SDK는 추적 ID를 생성하고 Lambda 함수에 대한 호출을 추적합니다. AWS Lambda 추적 데이터와 추적 ID를 Lambda 함수에 전달합니다. 또한 X-Ray SDK는 동일한 추적 ID를 사용하여 에이전트나 수집기에 데이터를 전송합니다. 따라서 API, AWS Lambda 서비스 및 Lambda 함수의 노드는 모두 별개이지만 연결된 노드로 추적 맵에 나타납니다.

서비스 그래프 데이터는 30일 동안 보관됩니다.

## 트레이스

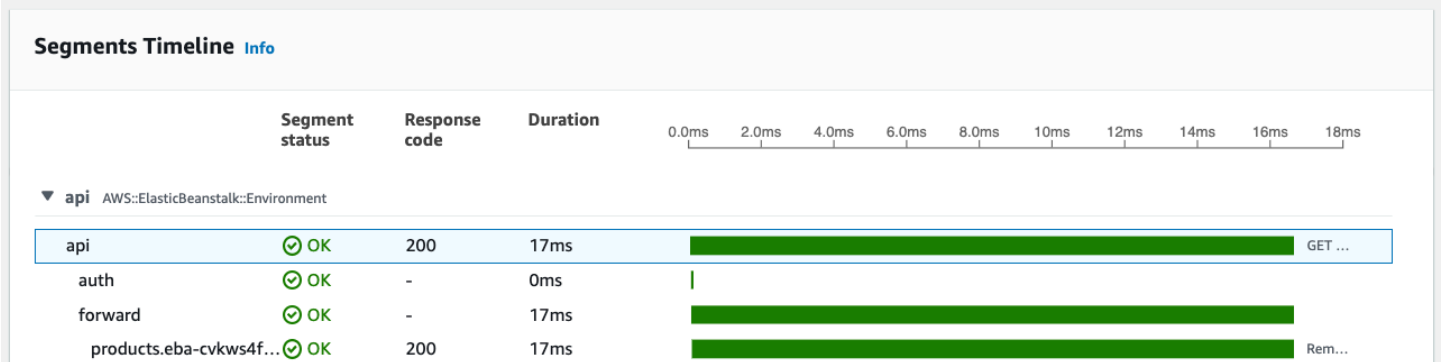
트레이스는 단일 요청에 의해 생성된 모든 세그먼트를 수집합니다. 트레이스는 [트레이스 ID](#)를 사용하여 애플리케이션을 통한 요청 경로를 추적합니다. 이 요청은 일반적으로 로드 밸런서를 통과하여 애플리케이션 코드와 상호 작용하고 다른 AWS 서비스 또는 외부 웹 API에 대한 다운스트림 호출을 생성하는 HTTP GET 또는 POST 요청입니다. HTTP 요청과 상호 작용하는 첫 번째 지원 서비스는 요청에 추적 ID 헤더를 추가합니다. 그러면 서비스가 추적 ID를 다운스트림으로 전파하여 지연 시간, 처리 및 기타 요청 데이터를 추적합니다.

다음 이미지는 요청을 처리하는 애플리케이션의 예를 보여줍니다. HTTP 추적 요약에는 HTTP 응답 코드, 요청 처리 시간, 애플리케이션이 요청을 처리한 시간이 포함됩니다. 다음 이미지는 각 추적 세그먼트의 타임라인도 보여줍니다. 타임라인에는 상태, HTTP 응답 코드, 세그먼트를 완료하는 데 걸린 시간이 표시됩니다. 차트는 트레이스에 있는 각 세그먼트의 지속 시간, 시작 및 종료 시간을 다른 세그먼트와 비교하여 보여줍니다.



**Trace Summary**

Method	Response Code	Duration	Age
GET	200	17ms	a few seconds (2022-01-20 16:35:56)



X-Ray Bills Trace 징수 방식에 대한 자세한 내용은 [AWS X-Ray 가격 책정에서](#) X-Ray Trace의 청구 방식에 대한 정보를 참조하십시오. 트레이스 데이터는 30일 동안 보관됩니다.

## 샘플링

X-Ray SDK는 샘플링 알고리즘을 적용하여 효율적인 추적을 보장하고 애플리케이션이 처리하는 요청의 대표적인 샘플을 제공합니다. 이 알고리즘은 추적되는 요청을 결정합니다. 기본적으로 X-Ray SDK는 1초가 시작될 때 수신된 첫 번째 요청과 모든 추가 요청의 5% 를 기록합니다.

시작할 때 서비스 요금이 발생하지 않도록 하려는 경우 기본 샘플링 비율은 보수적입니다. 기본 샘플링 비율을 변경하도록 X-Ray를 구성하고 서비스 또는 요청의 속성에 따라 샘플링을 적용하는 추가 규칙을 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 대용량 읽기 전용 통화의 경우.

자세한 내용은 [샘플링 규칙을 구성합니다.](#) 및 API를 참조하십시오. [CreateSamplingRule](#)

## 추적 헤더

모든 요청은 구성할 수 있는 최소 수까지 추적됩니다. 이 최소값에 도달하면 X-Ray는 요청의 일부만 추적하므로 추가 비용이 발생하지 않습니다. X-Ray는 로 시작하는 추적 헤더의 HTTP 요청에 샘플링 결정 및 추적 ID를 추가합니다. X-Amzn-Trace-Id 요청이 X-Ray와 통합된 첫 번째 AWS 서비스와 상호 작용할 때 X-Ray는 이러한 헤더를 추가합니다. X-Ray SDK는 이러한 헤더를 읽고 응답에 포함합니다.

Example 루트 트레이스 ID 및 샘플링 결정이 포함된 트레이스 헤더

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

### 트레이스 헤더 보안

추적 헤더는 X-Ray SDK AWS 서비스, 또는 클라이언트 요청에서 생성될 수 있습니다. 애플리케이션은 수신 요청에서 X-Amzn-Trace-Id를 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 계층 애플리케이션은 상위 세그먼트 ID를 사용하여 두 요청을 연결합니다.

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

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

Lambda 또는 AWS 서비스 다른 업체는 처리 메커니즘의 일부로 Lineage 시작하는 헤더의 일부를 추가할 수 있습니다. 추적 헤더의 추가된 부분을 직접 사용해서는 안 됩니다.

### Example 리니지를 이용한 헤더 추적

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

## 필터 표현식

소량의 데이터를 샘플링하더라도 복잡한 응용 프로그램에서는 많은 추적 데이터가 생성될 수 있습니다. [필터 표현식](#)을 사용하면 개별 요청, 특정 경로 또는 사용자에 대한 트레이스를 비롯한 특정 트레이스를 찾을 수 있습니다.

다음 이미지는 정의한 그룹을 기준으로 필터링하는 데 사용할 수 있는 X-Ray 콘솔의 텍스트 상자를 보여줍니다. 그룹에 대한 자세한 내용은 다음 섹션 그룹을 참조하십시오.

The screenshot shows the AWS X-Ray console interface. At the top, there are tabs for 'Traces' and 'Info', and a time range selector set to '5m'. Below this, there is a search bar with the text 'Filter by X-Ray group' and a query input field containing 'http.url CONTAINS "api/move/"'. A 'Run query' button is visible, along with a status indicator '5 traces retrieved'. Below the search bar, there is a section for 'Query refiners'. The main content area displays a table titled 'Traces (5)' with the following data:

ID	Trace status	Timestamp	Response code	Response Time	Duration	HTTP Method
...561513004630e58c75c992ed	OK	3.4min (2023-08-16 17:39:20)	200	0.104s	0.104s	POST
...2e83714b7daac593167d2e73	OK	3.4min (2023-08-16 17:39:19)	200	0.07s	0.07s	POST
...54740787431329383155f154	OK	3.4min (2023-08-16 17:39:18)	200	0.1s	0.1s	POST

## 그룹

필터 표현식 내의 그룹을 사용하여 추적 데이터의 양을 줄이고 그룹 기준에 맞는 데이터에 집중할 수 있습니다.

그룹을 사용하여 해당 그룹과 관련된 서비스 그래프, 추적 요약 및 CloudWatch 메트릭을 생성할 수 있습니다. 이름 또는 Amazon 리소스 이름 (ARN) 으로 호출할 수 있습니다. X-Ray는 들어오는 트레이스

가 X-Ray 서비스에 저장될 때 그룹 필터 표현식과 비교하여 확인합니다. CloudWatch 그룹 기준과 일치하는 트레이스에 대한 메트릭을 1분마다 게시합니다.

그룹의 필터 표현식을 업데이트해도 이미 기록한 데이터는 변경되지 않습니다. 업데이트는 후속 추적에만 적용됩니다. 이렇게 하면 새 표현식과 이전 표현식의 그래프를 병합할 수 있습니다. 연결되지 않은 그룹이 단일 그래프 내에 병합되지 않도록 하려면 현재 그룹을 삭제하고 [https://docs.aws.amazon.com/xray/latest/api/API\\_CreateGroup.html](https://docs.aws.amazon.com/xray/latest/api/API_CreateGroup.html) 새 그룹을 만드십시오.

### Note

그룹 요금은 검색된 트레이스 중 필터 표현식과 일치하는 트레이스 수를 기준으로 청구됩니다. 자세한 내용은 [AWS X-Ray 요금](#)을 참조하십시오.

그룹에 대한 자세한 정보는 [그룹 구성](#) 섹션을 참조하세요.

## 주석 및 메타데이터

애플리케이션을 계측할 때, X-Ray SDK는 수신 및 발신 요청에 대한 정보를 기록합니다. SDK는 사용된 AWS 리소스 및 애플리케이션 자체에 대한 정보도 기록합니다. 세그먼트 문서에 주석 및 메타데이터로 다른 정보를 추가할 수 있습니다. 주석과 메타데이터는 추적 수준에서 결합됩니다. 모든 세그먼트나 하위 세그먼트에 추가할 수 있습니다.

주석은 필터 [표현식에 사용하기 위해 인덱싱되는 키값 쌍입니다](#). 주석은 콘솔의 트레이스를 그룹화할 때 사용할 데이터를 기록하거나 [GetTraceSummaries](#) API를 호출할 때 사용합니다.

X-Ray가 트레이스당 최대 50개의 주석까지 인덱싱합니다.

메타데이터는 객체와 목록을 포함하여 인덱싱되지 않은 모든 유형의 값을 포함하는 키값 쌍입니다. 메타데이터를 사용하여 트레이스에 저장하고 싶지만 트레이스 검색에는 사용하지 않을 데이터를 기록합니다.

콘솔의 트레이스 세부 정보 페이지에 있는 세그먼트 또는 하위 세그먼트 세부 정보 창에서 주석과 메타데이터를 볼 수 있습니다. CloudWatch 자세한 내용은 [에서 추적 및 추적 세부 정보 보기](#)를 참조하십시오. [X-Ray 콘솔 살펴보기](#)

## 오류, 결합 및 예외

X-Ray는 애플리케이션 코드의 오류와 다운스트림 서비스에서 반환된 오류를 추적합니다. X-Ray는 요청에서 다음과 같은 HTTP 응답 코드를 추적합니다.

- **Error**— 클라이언트 오류 (400 시리즈 오류) 는 요청 자체에 오류가 있어 서버가 클라이언트의 요청을 이해하거나 처리할 수 없음을 나타냅니다. 이러한 오류는 구문 오류, 정보 누락 또는 잘못된 요청 본문 때문에 발생할 수 있습니다.
- **Fault**— 서버 장애 (500 시리즈 오류) 는 서버 자체에 문제가 있어 서버가 유효한 요청을 처리할 수 없었음을 나타냅니다. 이러한 오류는 소프트웨어 또는 하드웨어 장애 또는 서버의 리소스 제한과 같은 문제로 인해 발생할 수 있습니다.
- **Throttle**— 제한 오류 (429 Too Many Requests) 는 클라이언트가 일정 기간 동안 서버 또는 API 에 너무 많은 요청을 보낼 때 발생하는 특정 유형의 클라이언트 오류입니다.

애플리케이션이 계속된 요청을 처리하는 동안 예외가 발생하는 경우, X-Ray SDK는 스택 추적 ID (사용 가능한 경우) 를 포함하여 예외에 대한 세부 정보를 기록합니다. X-Ray 콘솔의 [세그먼트 세부 정보](#)에서 예외를 볼 수 있습니다.



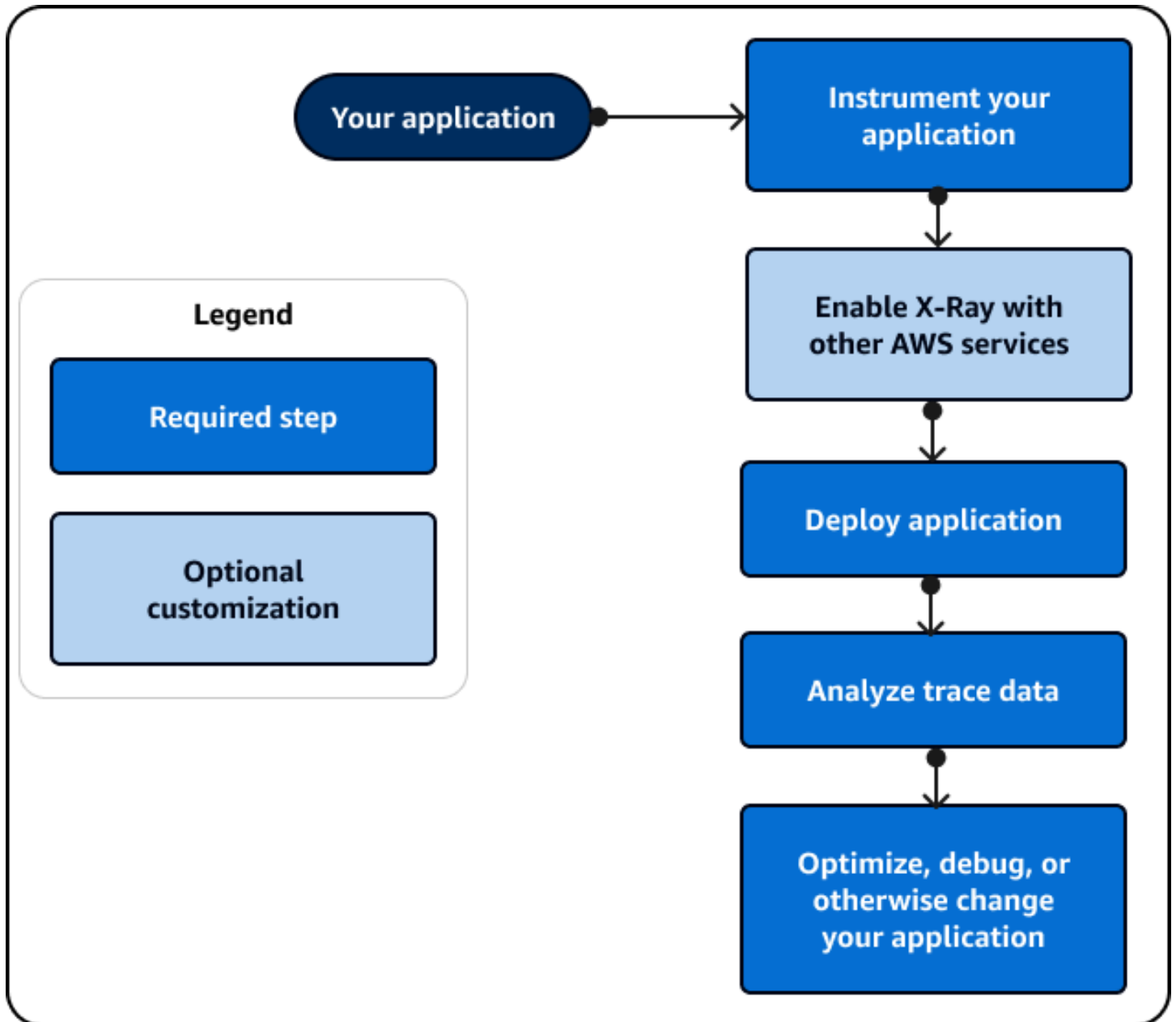
# X-Ray로 시작하세요

X-Ray를 사용하려면 다음을 수행해야 합니다.

1. 애플리케이션을 계측하면 X-Ray가 애플리케이션이 요청을 처리하는 방식을 추적할 수 있습니다.
  - X-Ray SDK, X-Ray API ADOT 또는 CloudWatch 애플리케이션 신호를 사용하여 트레이스 데이터를 X-Ray로 전송하십시오. 사용할 인터페이스에 대한 자세한 내용은 [인터페이스 선택](#)을 참조하십시오.

계측에 대한 자세한 내용은 [애플리케이션 측정 대상 AWS X-Ray](#)을 참조하십시오.
2. (선택 사항) X-Ray와 AWS 서비스 통합되는 다른 제품과 함께 작동하도록 X-Ray를 구성합니다. 트레이스를 샘플링하고 들어오는 요청에 헤더를 추가하고, 에이전트 또는 컬렉터를 실행하고, 트레이스 데이터를 자동으로 X-Ray에 보낼 수 있습니다. 자세한 정보는 [다른 AWS X-Ray 사람과 통합 AWS 서비스](#)을 참조하세요.
3. 계측 애플리케이션을 배포하십시오. 애플리케이션이 요청을 수신하면 X-Ray SDK가 추적, 세그먼트 및 서브세그먼트 데이터를 기록합니다. 이 단계에서 IAM 정책을 설정하고 에이전트 또는 컬렉터를 배포해야 할 수도 있습니다.
  - AWS Distro for OpenTelemetry (ADOT) SDK와 CloudWatch 에이전트를 사용하여 애플리케이션을 다른 플랫폼에 배포하기 위한 스크립트의 예는 [애플리케이션](#) 신호 데모 스크립트를 참조하십시오.
  - X-Ray SDK 및 X-Ray 데몬을 사용하여 애플리케이션을 배포하는 예제 스크립트는 [AWS X-Ray 샘플 애플리케이션](#)을 참조하십시오.
4. (선택 사항) 콘솔을 열어 데이터를 보고 분석합니다. 추적 맵, 서비스 맵 등의 GUI 표현을 보고 애플리케이션이 어떻게 작동하는지 검사할 수 있습니다. 콘솔의 그래픽 정보를 사용하여 애플리케이션을 최적화, 디버그 및 이해하십시오. 콘솔 선택에 대한 자세한 내용은 [사용 AWS Management Console](#)을 참조하십시오.

다음 다이어그램은 X-Ray를 사용하여 시작하는 방법을 보여줍니다.



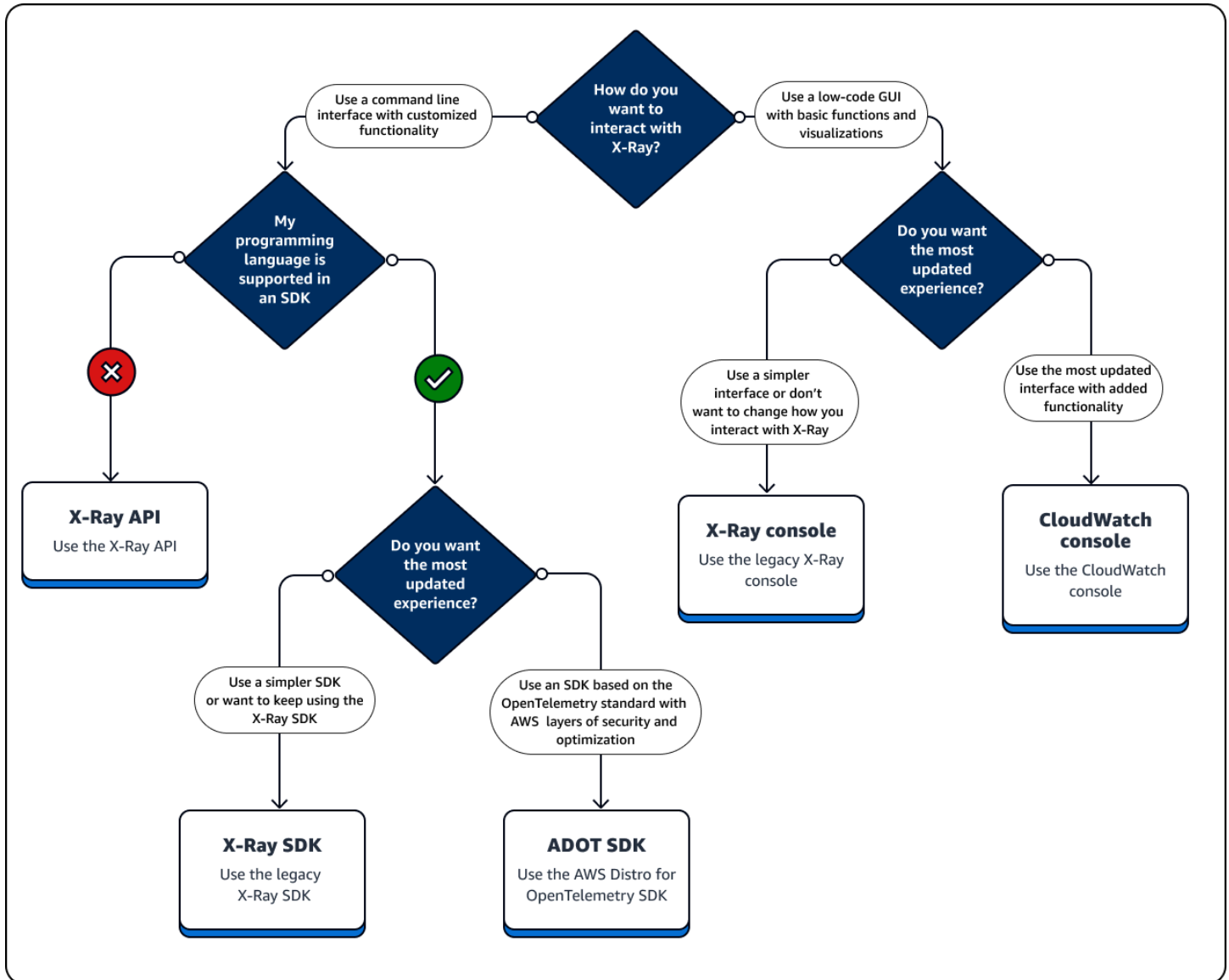
콘솔에서 사용할 수 있는 데이터 및 맵의 예를 보려면 추적 데이터를 생성하도록 이미 계측된 [샘플 애플리케이션](#)을 시작하십시오. 몇 분 안에 트래픽을 생성하고, 세그먼트를 X-Ray로 보내고, 추적 및 서비스 맵을 볼 수 있습니다.

## 인터페이스 선택

AWS X-Ray 애플리케이션이 작동하는 방식과 애플리케이션이 다른 서비스 및 리소스와 얼마나 잘 상호 작용하는지에 대한 통찰력을 제공할 수 있습니다. 애플리케이션을 계측하거나 구성한 후 애플리케이션이 요청을 처리할 때 X-Ray는 추적 데이터를 수집합니다. 이 추적 데이터를 분석하여 성능 문제를 식별하고, 오류를 해결하고, 리소스를 최적화할 수 있습니다. 이 안내서는 다음 지침에 따라 X-Ray와 상호 작용하는 방법을 보여줍니다.

- 빠르게 시작하거나 사전 구축된 시각화를 사용하여 기본 작업을 수행할 수 있는 AWS Management Console 경우에는 a를 사용하십시오.
  - 모든 X-Ray CloudWatch 콘솔 기능이 포함된 가장 업데이트된 사용자 경험을 제공하는 Amazon 콘솔을 선택하십시오.
  - 더 간단한 인터페이스를 원하거나 X-Ray와 상호 작용하는 방식을 바꾸고 싶지 않다면 X-Ray 콘솔을 사용하세요.
- 제공할 AWS Management Console 수 있는 것보다 더 많은 사용자 지정 추적, 모니터링 또는 로깅 기능이 필요한 경우 SDK를 사용하세요.
  - 보안 및 최적화 계층이 추가된 오픈소스 ADOT OpenTelemetry SDK를 기반으로 벤더에 구매받지 않는 SDK를 원한다면 SDK를 선택하세요. AWS
  - 더 간단한 SDK를 원하거나 애플리케이션 코드를 업데이트하고 싶지 않다면 X-Ray SDK를 선택하세요.
- SDK가 애플리케이션의 프로그래밍 언어를 지원하지 않는 경우 X-Ray API 작업을 사용하십시오.

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



### 인터페이스 유형 살펴보기

- [사용 AWS Management Console](#)
- [SDK 사용](#)
- [X-Ray API 사용](#)

## 사용 AWS Management Console

최소한의 코딩이 필요한 그래픽 사용자 인터페이스 (GUI) 를 AWS Management Console 원한다면 를 사용하십시오. X-Ray를 처음 사용하는 사용자는 사전 구축된 시각화를 사용하고 기본 작업을 수행하여 빠르게 시작할 수 있습니다. 콘솔에서 직접 다음 작업을 수행할 수 있습니다.

- X-Ray 활성화하기
- 애플리케이션 성능에 대한 개괄적인 요약은 볼 수 있습니다.
- 애플리케이션 상태를 확인하세요.
- 높은 수준의 오류를 식별하십시오.
- 기본 추적 요약은 확인하세요.

<https://console.aws.amazon.com/cloudwatch/> 의 Amazon CloudWatch 콘솔 또는 <https://console.aws.amazon.com/xray/home> 의 X-Ray 콘솔 중 하나를 사용하여 X-Ray와 상호 작용할 수 있습니다.

## Amazon CloudWatch 콘솔 사용

CloudWatch 콘솔에는 사용하기 쉽도록 X-Ray 콘솔에서 재설계된 새로운 X-Ray 기능이 포함되어 있습니다. CloudWatch 콘솔을 사용하는 경우 X-Ray Trace 데이터와 함께 CloudWatch 로그 및 메트릭을 볼 수 있습니다. CloudWatch 콘솔을 사용하여 다음을 포함한 데이터를 보고 분석할 수 있습니다.

- X-Ray trace — 요청을 처리할 때 애플리케이션과 관련된 트레이스를 보고, 분석하고, 필터링합니다. 이러한 추적을 사용하여 높은 지연 시간을 찾고, 오류를 디버깅하고, 애플리케이션 워크플로를 최적화하십시오. 추적 맵과 서비스 맵을 보고 애플리케이션 워크플로를 시각적으로 확인할 수 있습니다.
- 로그 — 애플리케이션에서 생성되는 로그를 보고, 분석하고, 필터링합니다. 로그를 사용하여 오류를 해결하고 특정 로그 값을 기반으로 모니터링을 설정합니다.
- 지표 — 리소스가 내보내는 지표를 사용하여 애플리케이션 성능을 측정 및 모니터링하거나 자체 지표를 생성합니다. 이러한 지표를 그래프와 차트로 볼 수 있습니다.
- 네트워크 및 인프라 모니터링 — 주요 네트워크의 정전 여부와 컨테이너식 애플리케이션, 기타 AWS 서비스, 클라이언트를 포함한 인프라의 상태 및 성능을 모니터링합니다.
- 다음 X-Ray 콘솔 사용 섹션에 나열된 X-Ray 콘솔의 모든 기능

CloudWatch 콘솔에 대한 자세한 내용은 [Amazon 시작하기](#)를 참조하십시오 CloudWatch.

<https://console.aws.amazon.com/cloudwatch/> 에서 아마존 CloudWatch 콘솔에 로그인합니다.

## X-Ray 콘솔 사용하기

X-Ray 콘솔은 애플리케이션 요청에 대한 분산 추적 기능을 제공합니다. 보다 간단한 콘솔 환경을 원하거나 애플리케이션 코드를 업데이트하고 싶지 않은 경우 X-Ray 콘솔을 사용하십시오. AWS 더 이상 X-

Ray 콘솔을 개발하지 않습니다. X-Ray 콘솔에는 계속 애플리케이션을 위한 다음과 같은 기능이 포함되어 있습니다.

- [인사이트](#) — 애플리케이션 성능의 이상을 자동으로 감지하고 근본 원인을 찾아냅니다. 인사이트는 CloudWatch 콘솔의 Insights 아래에 포함됩니다. 자세한 내용은 X-Ray Insights 사용을 참조하십시오 [X-Ray 콘솔 살펴보기](#).
- 서비스 맵 — 애플리케이션의 그래픽 구조 및 클라이언트, 리소스, 서비스 및 종속성과의 연결을 그래프로 표시합니다.
- 트레이스 — 애플리케이션이 요청을 처리할 때 생성되는 트레이스의 개요를 확인하세요. 추적 데이터를 사용하여 HTTP 응답 및 응답 시간을 비롯한 기본 지표에 대한 애플리케이션 성능을 파악할 수 있습니다.
- 분석 — 응답 시간 분포를 위한 그래프를 사용하여 추적 데이터를 해석, 탐색 및 분석합니다.
- 구성 — 사용자 지정 트레이스를 생성하여 다음에 대한 기본 구성을 변경합니다.
  - 샘플링 — 애플리케이션에서 추적 정보를 샘플링하는 빈도를 정의하는 규칙을 만드십시오. 자세한 내용은 [에서 샘플링 규칙 구성을 참조하십시오](#) [X-Ray 콘솔 살펴보기](#).
  - [암호화](#) - 감사하거나 사용하지 않도록 설정할 수 있는 키를 사용하여 저장된 데이터를 암호화합니다. AWS Key Management Service
  - 그룹 - 필터 표현식을 사용하여 URL 이름 또는 응답 시간과 같은 공통 기능을 가진 추적 그룹을 정의합니다. 자세한 내용은 [그룹 구성을](#) 참조하십시오.

<https://console.aws.amazon.com/xray/home> 에서 X-Ray 콘솔에 로그인합니다.

## X-Ray 콘솔 살펴보기

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

### Note

X-Ray Service CloudWatch ServiceLens 맵과 맵이 Amazon CloudWatch 콘솔 내의 X-Ray 추적 맵에 통합되었습니다. [CloudWatch 콘솔을](#) 열고 왼쪽 탐색 창의 X-Ray 트레이스에서 Trace Map을 선택합니다.

CloudWatch 이제 [애플리케이션 서비스, 클라이언트, Synthetics canary 및 서비스 종속성을 검색하고 모니터링할 수 있는 애플리케이션 시그널](#)이 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 생성된 추적 데이터로부터 X-Ray가 생성하는 JSON 서비스 그래프를 시각적으로 표현한 추적 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 섹션에서 X-Ray의 콘솔 환경을 보고 CloudWatch 콘솔과 비교해 보십시오.

## X-Ray 트레이스 맵 사용

X-Ray trace 맵을 보고 오류가 발생한 서비스, 대기 시간이 긴 연결 또는 실패한 요청에 대한 추적을 식별할 수 있습니다.

### Note

CloudWatch 이제 [애플리케이션 서비스, 클라이언트, 합성 카나리아 및 서비스 종속성을 검색하고 모니터링할 수 있는 애플리케이션 시그널](#)이 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 추적 맵에 통합됩니다. [CloudWatch 콘솔](#)을 열고 왼쪽 탐색 창의 X-Ray 트레이스에서 Trace Map을 선택합니다.

## 추적 맵 보기

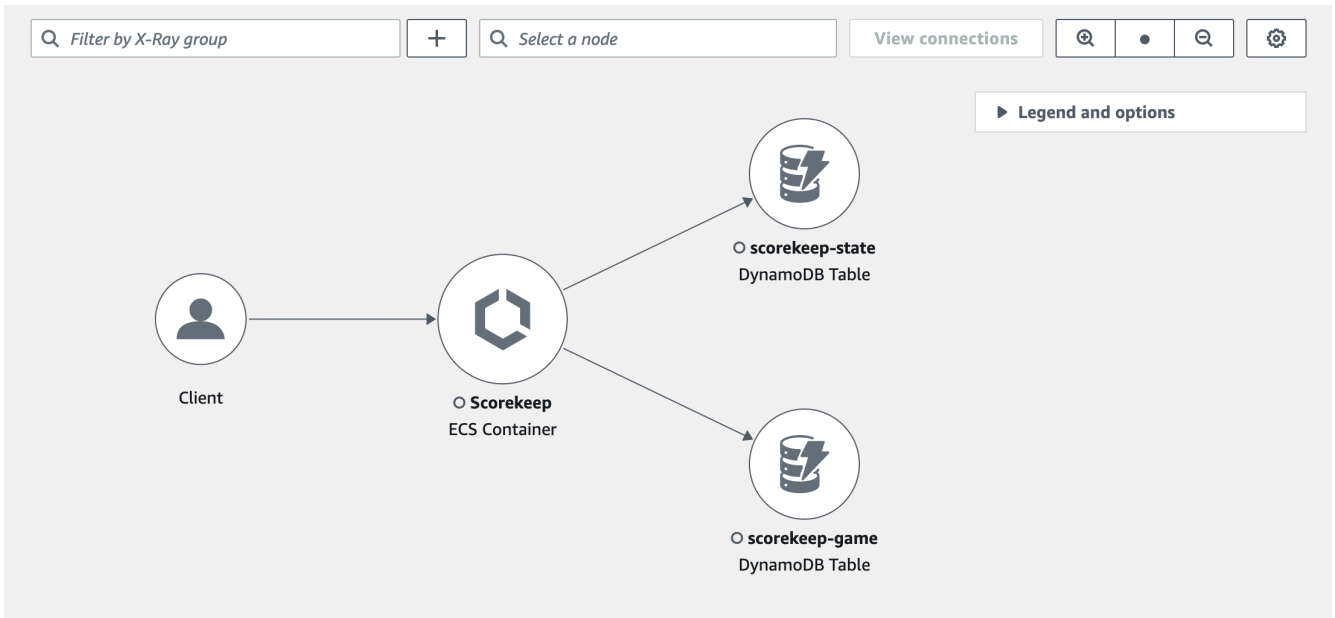
트레이스 맵은 애플리케이션에서 생성된 트레이스 데이터를 시각적으로 표현한 것입니다. 맵에는 요청을 처리하는 서비스 노드, 요청의 출처를 나타내는 업스트림 클라이언트 노드, 요청을 처리하는 동안 애플리케이션에서 사용하는 웹 서비스 및 리소스를 나타내는 다운스트림 서비스 노드가 표시됩니다.

추적 맵은 Amazon SQS 및 Lambda를 사용하는 이벤트 기반 애플리케이션 전반의 추적에 대한 연결된 보기를 표시합니다. [자세한 내용은 다음 Trace 이벤트 기반 애플리케이션 섹션을 참조하십시오.](#) 추적 맵은 [계정 간 추적도](#) 지원하여 여러 계정의 노드를 단일 맵에 표시합니다.

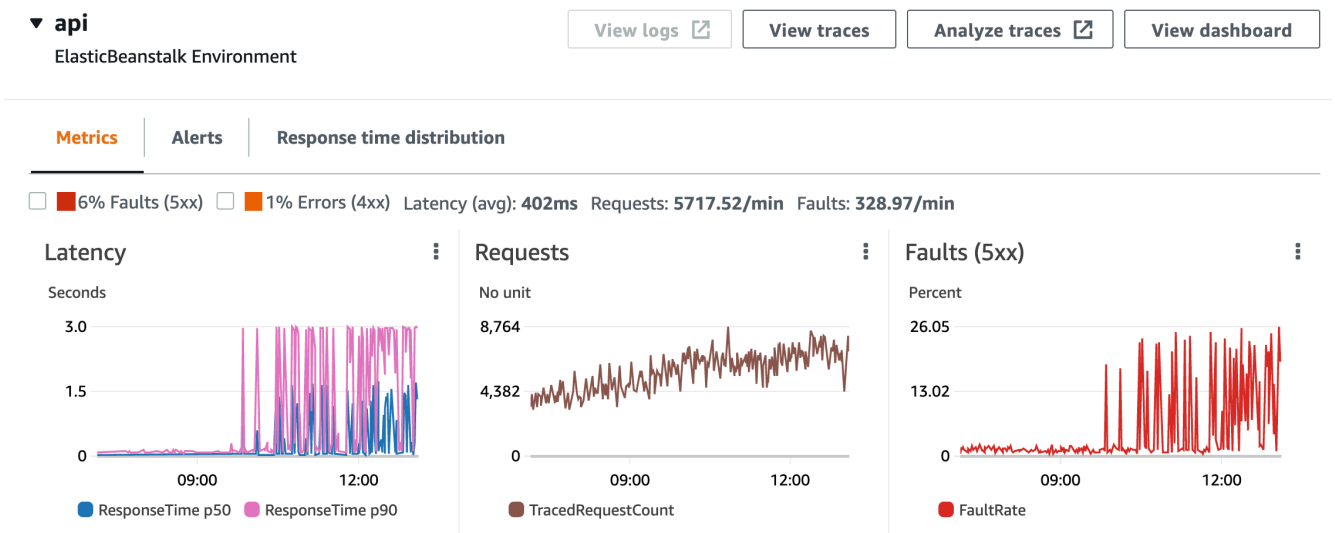
## CloudWatch console

콘솔에서 트레이스 맵을 보려면 CloudWatch

1. [CloudWatch 콘솔](#)을 엽니다. 왼쪽 탐색 창의 X-Ray Traces 섹션에서 Trace Map을 선택합니다.



2. 노드에 대한 요청을 보려는 서비스 노드 또는 그 사이를 이동하는 요청을 보려는 두 노드 간 에지를 선택합니다.
3. 지표, 알림, 응답 시간 분포 탭을 비롯한 추가 정보가 트레이스 맵 아래에 표시됩니다. Metrics 탭에서 각 그래프 내의 범위를 선택하여 자세히 살펴보거나 Fault 또는 Error 옵션을 선택하여 트레이스를 필터링할 수 있습니다. 응답 시간 분포 탭에서 그래프 내의 범위를 선택하여 응답 시간별로 추적을 필터링합니다.





- 트레이스 보기를 선택하여 트레이스를 보거나, 필터가 적용된 경우 필터링된 트레이스 보기를 선택합니다.
- 선택한 노드와 관련된 CloudWatch 로그를 보려면 로그 보기를 선택합니다. 모든 트레이스 맵 노드가 로그 보기를 지원하는 것은 아닙니다. 자세한 내용은 [문제 해결 CloudWatch 로그를 참조](#)하십시오.

추적 맵은 각 노드의 문제를 색상으로 윤곽선으로 표시하여 나타냅니다.

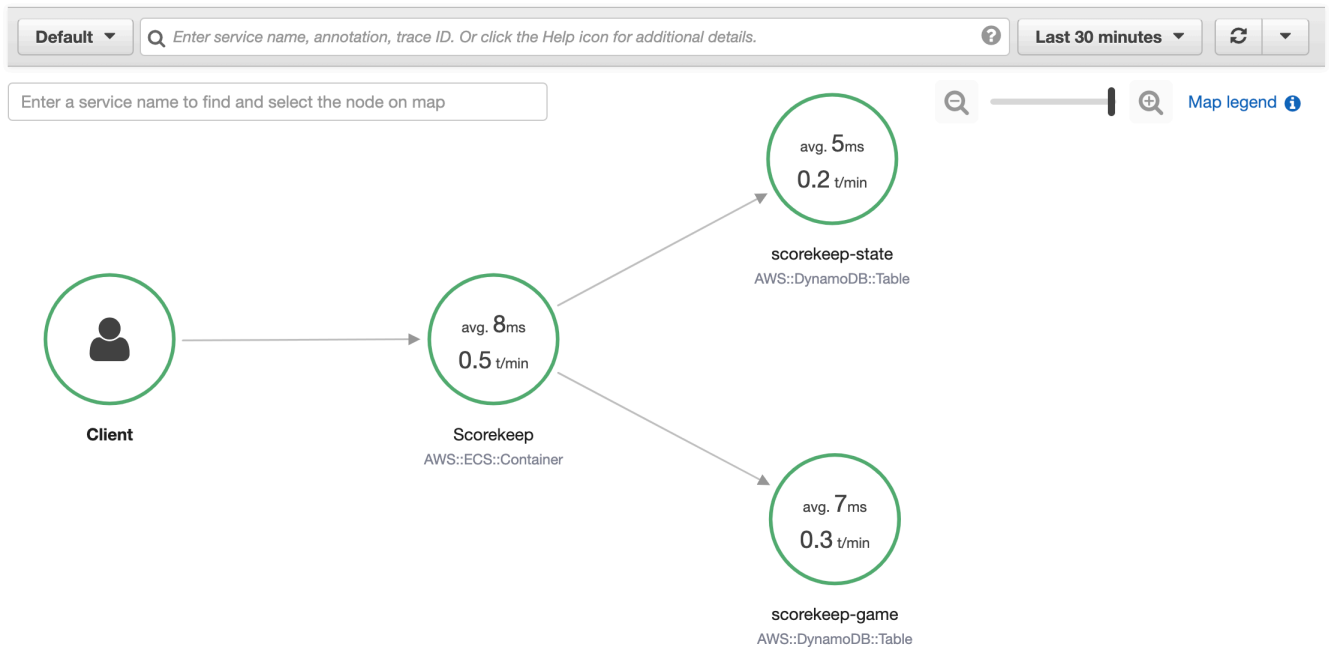
- 빨간색은 서버 장애를 의미합니다(500 시리즈 오류)
- 노란색은 클라이언트 오류를 의미합니다(400 시리즈 오류)
- 보라색은 병목 오류를 의미합니다(429 요청 과다)

트레이스 맵이 크면 화면 컨트롤이나 마우스를 사용하여 맵을 확대 및 축소하고 이리저리 이동할 수 있습니다.

## X-Ray console

### 서비스 맵을 보려면

- [X-Ray 콘솔](#)을 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. 분포 히스토그램에 대한 자세한 내용은 [여기](#)를 참조하십시오. ???

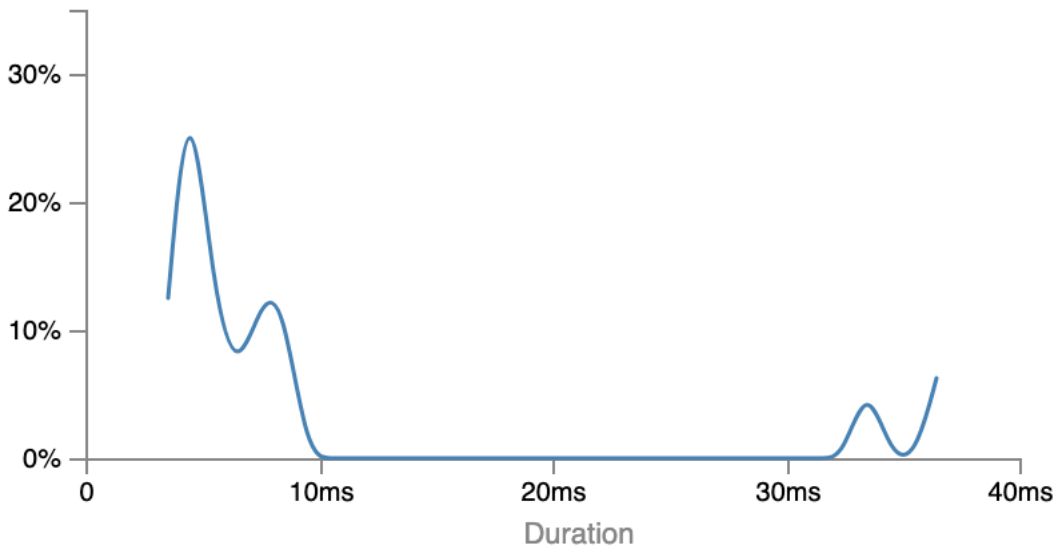
### Service details ?

**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.



### 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

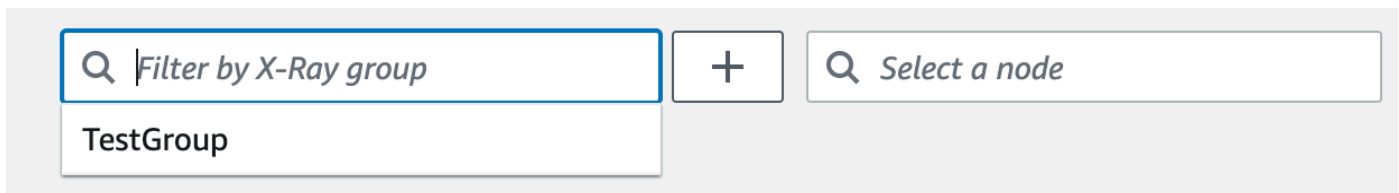
X-Ray 추적 맵은 최대 10,000개의 노드를 표시할 수 있습니다. 드문 경우지만 총 서비스 노드 수가 이 제한을 초과하는 경우 오류가 발생하여 콘솔에 전체 추적 맵을 표시하지 못할 수 있습니다.

## 그룹별 트레이스 맵 필터링

필터 표현식을 사용하여 그룹에 트레이스를 허용하는 기준을 정의할 수 있습니다. 필터 표현식에 대한 자세한 내용은 필터 [표현식 사용](#)을 참조하십시오. 그런 다음 다음 단계를 사용하여 추적 맵에 해당 특정 그룹을 표시하십시오.

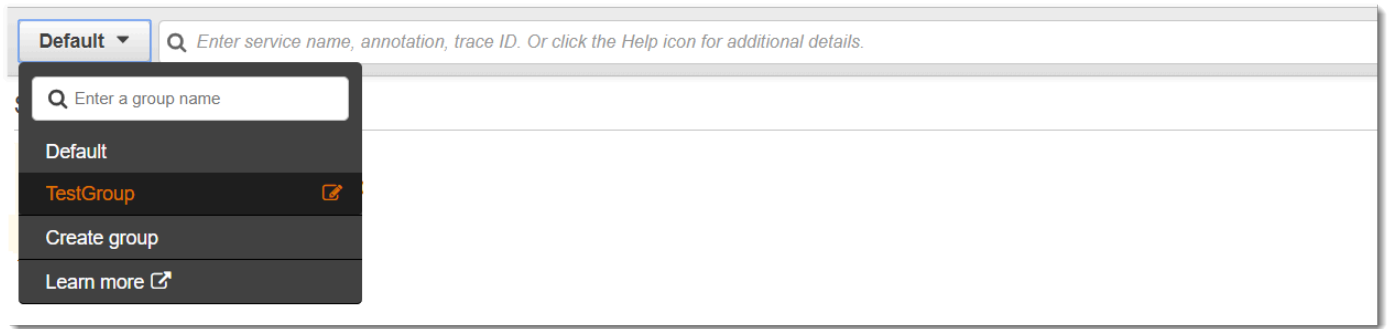
### CloudWatch console

추적 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

추적 맵에는 범례와 맵 표시를 사용자 지정하기 위한 여러 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. 트레이스 맵의 오른쪽 하단에서 다음을 포함한 서비스 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. 추적 목록에서 트레이스를 선택하면 트레이스 세부 정보 페이지에 선택한 트레이스와 관련된 서비스 노드의 맵과 트레이스 세그먼트의 타임라인이 표시됩니다.

### 추적 보기

#### CloudWatch console

CloudWatch 콘솔에서 트레이스를 보려면


1. <https://console.aws.amazon.com/cloudwatch/> 에서 AWS Management Console 로그인하고 CloudWatch 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 X-Ray-traces를 선택한 다음 Traces를 선택합니다. 그룹별로 필터링하거나 페이지 하단의 트레이스 섹션에 표시되는 트레이스를 필터링하는 필터 표현식을 입력할 수 있습니다. 자세한 내용은 [필터 표현식 사용](#)을 참조하십시오.

또는 서비스 맵을 사용하여 특정 서비스 노드로 이동한 다음 트레이스를 볼 수 있습니다. 그러면 쿼리가 이미 적용된 Traces 페이지가 열립니다.

3. 쿼리 세분화 섹션에서 쿼리를 구체화하세요. 트레이스를 공통 속성으로 필터링하려면 쿼리 세분화 기준 옆의 아래쪽 화살표에서 옵션을 선택합니다. 옵션에는 다음 사항이 포함됩니다.
  - 노드 - 트레이스를 서비스 노드별로 필터링합니다.
  - 리소스 ARN — 추적과 연결된 리소스를 기준으로 추적을 필터링합니다. 이러한 리소스의 예로는 Amazon Elastic Compute Cloud (Amazon EC2) AWS Lambda 인스턴스, 함수 또는 테이블이 있습니다. Amazon DynamoDB
  - 사용자 — 사용자 ID로 트레이스를 필터링합니다.
  - 오류 근본 원인 메시지 - 오류 근본 원인별로 트레이스를 필터링합니다.
  - URL - 애플리케이션에서 사용하는 URL 경로를 기준으로 트레이스를 필터링합니다.
  - HTTP 상태 코드 - 애플리케이션에서 반환한 HTTP 상태 코드를 기준으로 트레이스를 필터링합니다. 사용자 지정 응답 코드를 지정하거나 다음 중에서 선택할 수 있습니다.
    - 200— 요청이 성공했습니다.
    - 401— 요청에 유효한 인증 자격 증명이 부족했습니다.
    - 403— 요청에 유효한 권한이 없습니다.
    - 404— 서버가 요청된 리소스를 찾을 수 없습니다.
    - 500— 서버에 예상치 못한 상황이 발생하여 내부 오류가 발생했습니다.

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

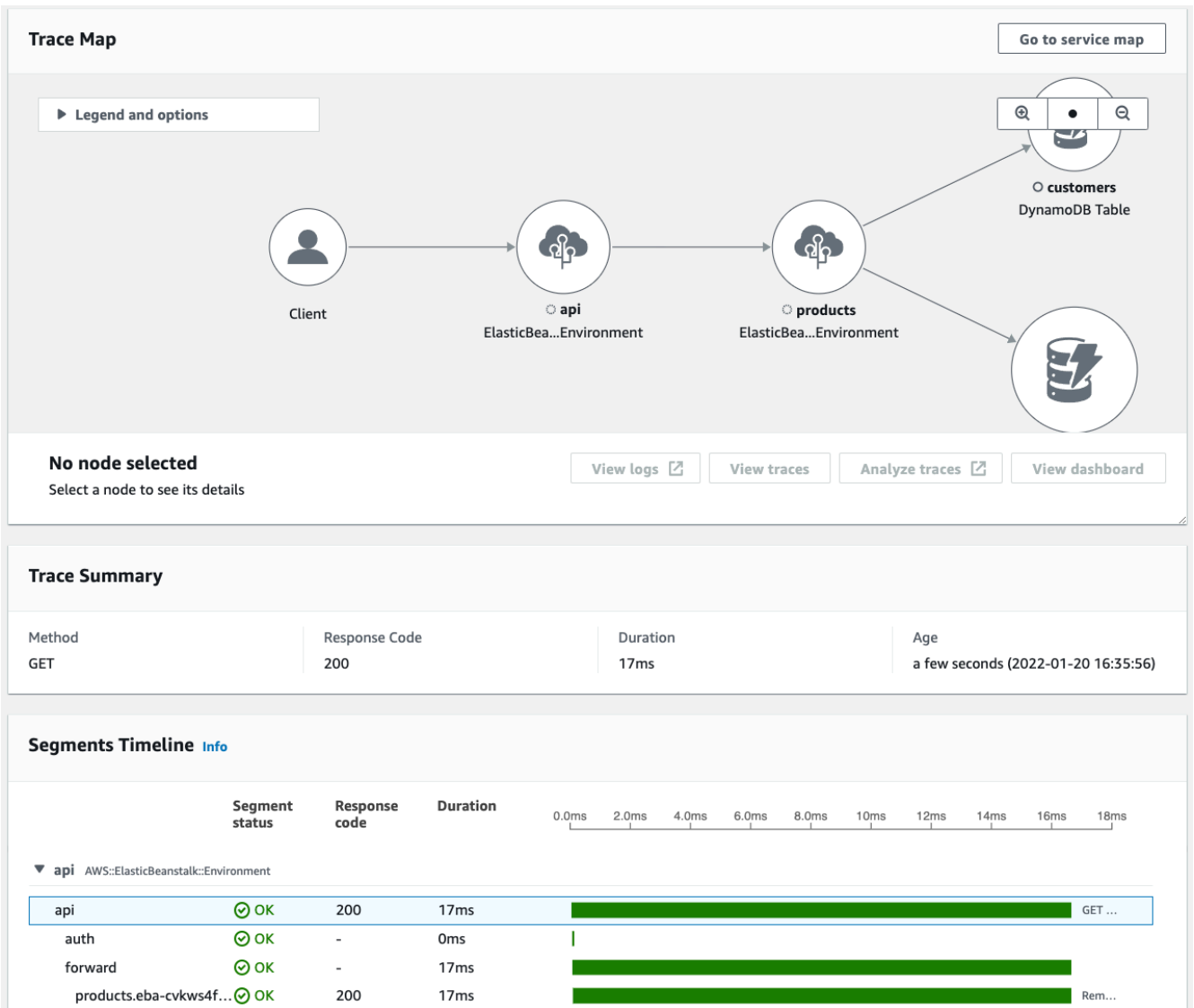
4. 단일 트레이스를 찾으려면 쿼리 필드에 [트레이스 ID](#)를 직접 입력합니다. X-Ray 형식 또는 월드 와이드 웹 컨소시엄 (W3C) 형식을 사용할 수 있습니다. 예를 들어 [AWS 배포판을](#) 사용하여 만든 트레이스는 W3C 형식입니다. OpenTelemetry

 Note

W3C 형식 추적 ID로 만든 트레이스를 쿼리하면 콘솔에 일치하는 트레이스가 X-Ray 형식으로 표시됩니다. 예를 들어 W3C 형식으로 쿼리하면 4efaaf4d1e8720b39541901950019ee5 콘솔에 해당하는 X-Ray가 표시됩니다. 1-4efaaf4d-1e8720b39541901950019ee5

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

다음 이미지는 트레이스와 관련된 서비스 노드가 포함된 트레이스 맵과 트레이스를 구성하는 세그먼트가 취한 경로를 나타내는 노드 사이의 에지를 보여줍니다. 트레이스 요약은 트레이스 맵 뒤에 표시됩니다. 요약에는 샘플 **GET** 작업, 응답 코드, 추적 실행 기간, 요청 기간에 대한 정보가 포함됩니다. 세그먼트 타임라인은 추적 세그먼트와 하위 세그먼트의 기간을 보여주는 추적 요약을 따릅니다.



Amazon SQS와 Lambda를 사용하는 이벤트 기반 애플리케이션이 있는 경우, Trace 맵에서 각 요청에 대한 추적의 연결된 보기를 볼 수 있습니다. 맵에서 메시지 생산자의 추적은 AWS Lambda 소비자의 추적에 연결되며 점선 가장자리로 표시됩니다. 이벤트 기반 애플리케이션에 대한 자세한 내용은 [이벤트 기반 애플리케이션을 추적합니다.](#)

추적 및 추적 세부 정보 페이지는 추적 목록과 단일 추적 맵 내에 여러 계정의 흔적을 나열할 수 있는 교차 계정 추적도 지원합니다. 자세한 정보는 [교차 계정 추적](#)을 참조하세요.



## X-Ray console

X-Ray 콘솔에서 트레이스를 보려면

1. X-Ray 콘솔의 [트레이스](#) 페이지를 엽니다. 추적 개요 패널에는 오류 근본 원인, ResourceARN 등을 비롯한 일반적인 기능별로 그룹화된 추적 목록이 표시됩니다. InstanceId
2. 그룹화된 추적 세트를 볼 수 있는 공통 기능을 선택하려면 그룹화 기준 옆의 아래쪽 화살표를 펼치십시오. 다음 그림은 의 URL별로 그룹화된 트레이스의 트레이스 개요와 관련 트레이스 목록을 보여줍니다. [AWS X-Ray 샘플 애플리케이션](#)

Trace overview

Group by:

URL	Avg response time	% of Traces	Response
<a href="http://scorekeep.elasticbeanstalk.com/api/user">http://scorekeep.elasticbeanstalk.com/api/user</a>	391 ms	4.76%	1 OK, 0 Throttled, 0 Errors, 0 Faults
<a href="http://scorekeep.elasticbeanstalk.com/api/session/8N63LUQ6">http://scorekeep.elasticbeanstalk.com/api/session/8N63LUQ6</a>	33.0 ms	4.76%	1 OK, 0 Throttled, 0 Errors, 0 Faults
<a href="http://scorekeep.elasticbeanstalk.com/api/session">http://scorekeep.elasticbeanstalk.com/api/session</a>	90.5 ms	9.52%	2 OK, 0 Throttled, 0 Errors, 0 Faults

Trace list (21)

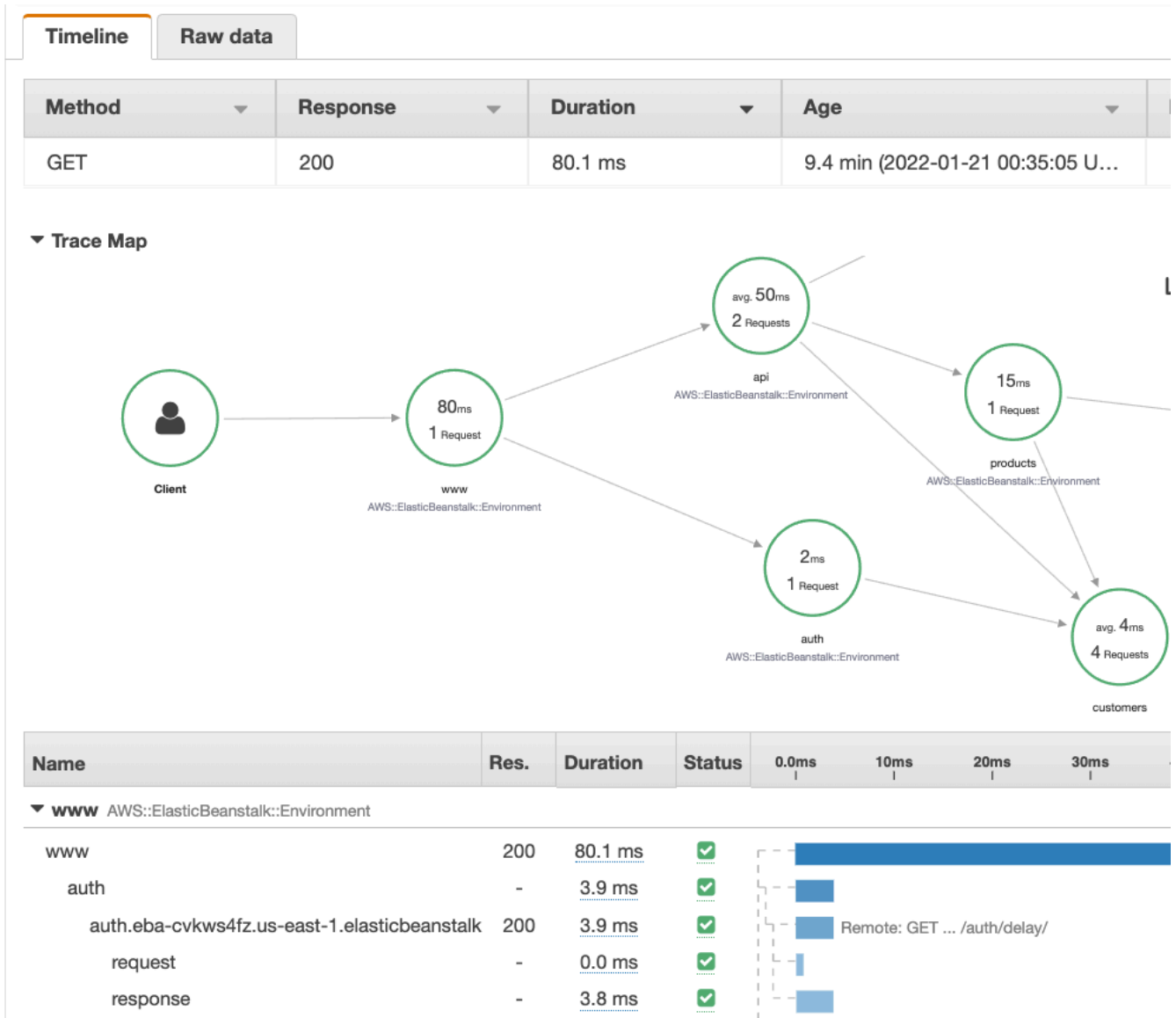
ID	Age	Method	Response	Response time	URL	Annotations
...f5f2df73	5.0 min	POST	200	391 ms	<a href="http://scorekeep.elasticbeanstalk.com/api/user">http://scorekeep.elasticbeanstalk.com/api/user</a>	0
...cfe39980	5.0 min	PUT	200	33.0 ms	<a href="http://scorekeep.elasticbeanstalk.com/api/session/8N63LUQ6">http://scorekeep.elasticbeanstalk.com/api/session/8N63LUQ6</a>	0
...dd653e4c	5.0 min	POST	200	19.0 ms	<a href="http://scorekeep.elasticbeanstalk.com/api/session">http://scorekeep.elasticbeanstalk.com/api/session</a>	0
...4765fec8	5.0 min	GET	200	162 ms	<a href="http://scorekeep.elasticbeanstalk.com/api/session">http://scorekeep.elasticbeanstalk.com/api/session</a>	0
...84eeef29	4.7 min	POST	200	95.0 ms	<a href="http://scorekeep.elasticbeanstalk.com/api/move/8N63LUQ6/2N56AC7L/PPMPBLJB">http://scorekeep.elasticbeanstalk.com/api/move/8N63LUQ6/2N56AC7L/PPMPBLJB</a>	1
...3ab33fdb	4.8 min	POST	200	95.0 ms	<a href="http://scorekeep.elasticbeanstalk.com/api/move/8N63LUQ6/2N56AC7L/PPMPBLJB">http://scorekeep.elasticbeanstalk.com/api/move/8N63LUQ6/2N56AC7L/PPMPBLJB</a>	1
...237e0705	4.8 min	POST	200	295 ms	<a href="http://scorekeep.elasticbeanstalk.com/api/move/8N63LUQ6/2N56AC7L/PPMPBLJB">http://scorekeep.elasticbeanstalk.com/api/move/8N63LUQ6/2N56AC7L/PPMPBLJB</a>	1
...86782227	4.9 min	POST	200	25.0 ms	<a href="http://scorekeep.elasticbeanstalk.com/api/game/8N63LUQ6/2N56AC7L/users">http://scorekeep.elasticbeanstalk.com/api/game/8N63LUQ6/2N56AC7L/users</a>	1
...fd82cc32	4.9 min	PUT	200	121 ms	<a href="http://scorekeep.elasticbeanstalk.com/api/game/8N63LUQ6/2N56AC7L/rules/TicTacToe">http://scorekeep.elasticbeanstalk.com/api/game/8N63LUQ6/2N56AC7L/rules/TicTacToe</a>	1
...7ca2e05f	1.4 min	GET	200	14.0 ms	<a href="http://scorekeep.elasticbeanstalk.com/api/game/8N63LUQ6/2N56AC7L">http://scorekeep.elasticbeanstalk.com/api/game/8N63LUQ6/2N56AC7L</a>	0
...062ccac5	1.7 min	GET	200	12.0 ms	<a href="http://scorekeep.elasticbeanstalk.com/api/game/8N63LUQ6/2N56AC7L">http://scorekeep.elasticbeanstalk.com/api/game/8N63LUQ6/2N56AC7L</a>	0
...dc0ebe3c	1.9 min	GET	200	9.0 ms	<a href="http://scorekeep.elasticbeanstalk.com/api/game/8N63LUQ6/2N56AC7L">http://scorekeep.elasticbeanstalk.com/api/game/8N63LUQ6/2N56AC7L</a>	0
...524637dc	4.9 min	PUT	200	69.0 ms	<a href="http://scorekeep.elasticbeanstalk.com/api/game/8N63LUQ6/2N56AC7L">http://scorekeep.elasticbeanstalk.com/api/game/8N63LUQ6/2N56AC7L</a>	1
...fdf5bb67	4.9 min	POST	200	81.0 ms	<a href="http://scorekeep.elasticbeanstalk.com/api/game/8N63LUQ6">http://scorekeep.elasticbeanstalk.com/api/game/8N63LUQ6</a>	1

3. 트레이스 목록에서 트레이스를 보려면 트레이스 ID를 선택하세요. 탐색 창에서 서비스 맵을 선택하여 특정 서비스 노드에 대한 트레이스를 볼 수도 있습니다. 그러면 해당 노드와 관련된 트레이스를 볼 수 있습니다.

타임라인 탭에는 추적에 대한 요청 흐름이 표시되며, 여기에는 다음이 포함됩니다.

- 트레이스에 있는 각 세그먼트의 경로 맵.
- 세그먼트가 트레이스 맵의 노드에 도달하는 데 걸린 시간
- 추적 맵의 노드에 대한 요청 횟수

다음 그림은 샘플 애플리케이션에 대한 GET 요청과 관련된 예제 Trace Map을 보여줍니다. 화살표는 각 세그먼트가 요청을 완료하는 데 걸린 경로를 보여줍니다. 서비스 노드는 요청 중에 이루어진 요청 수를 보여줍니다. GET



타임라인 탭에 대한 자세한 내용은 다음 추적 타임라인 살펴보기 섹션을 참조하십시오.

원시 데이터 탭에는 트레이스와 트레이스를 구성하는 세그먼트 및 하위 세그먼트에 대한 정보가 형식으로 표시됩니다. JSON 이 정보에는 다음이 포함될 수 있습니다.

- 타임스탬프
- 고유 ID

- 세그먼트 또는 하위 세그먼트와 관련된 리소스
- 세그먼트 또는 하위 세그먼트의 출처 또는 출처
- 애플리케이션에 대한 요청에 대한 추가 정보 (예: HTTP 요청의 응답)

## 트레이스 타임라인 살펴보기

타임라인 섹션에는 작업을 완료하는 데 사용한 시간에 해당하는 수평 막대 옆에 세그먼트와 하위 세그먼트의 계층 구조가 표시됩니다. 목록의 첫 번째 항목이 세그먼트인데, 단일 요청의 서비스에 의해 기록된 모든 데이터를 나타냅니다. 하위 세그먼트는 세그먼트 다음에 들여쓰여지고 나열됩니다. 옆에는 각 세그먼트에 대한 정보가 포함됩니다.

## CloudWatch console

CloudWatch 콘솔의 세그먼트 타임라인은 다음 정보를 제공합니다.

- 첫 번째 열: 선택한 트레이스의 세그먼트와 하위 세그먼트를 나열합니다.
- 세그먼트 상태 열: 각 세그먼트와 하위 세그먼트의 상태 결과를 나열합니다.
- 응답 코드 열: 세그먼트나 하위 세그먼트에서 이루어진 브라우저 요청에 대한 HTTP 응답 상태 코드가 있는 경우 해당 코드를 나열합니다.
- 기간 열: 세그먼트 또는 하위 세그먼트가 실행된 시간을 나열합니다.
- Hosted in 열: 해당하는 경우 세그먼트 또는 하위 세그먼트가 실행되는 네임스페이스나 환경을 나열합니다. 자세한 정보는 <https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/AppSignals-StandardMetrics.html#AppSignals-StandardMetrics-Dimensions>을 참조하세요.
- 마지막 열: 타임라인의 다른 세그먼트 또는 하위 세그먼트와 비교하여 세그먼트 또는 하위 세그먼트가 실행된 기간에 해당하는 가로 막대를 표시합니다.

세그먼트와 하위 세그먼트 목록을 서비스 노드별로 그룹화하려면 노드별 그룹화를 켜십시오.

## X-Ray console

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

X-Ray 콘솔의 타임라인은 다음 정보를 제공합니다.

- 이름 열: 트레이스에 있는 세그먼트와 하위 세그먼트의 이름을 나열합니다.

- **Res. 열:** 세그먼트나 하위 세그먼트의 브라우저 요청에 대한 HTTP 응답 상태 코드가 있는 경우 해당 세그먼트나 하위 세그먼트에서 이루어진 브라우저 요청에 대한 HTTP 응답 상태 코드를 나열합니다.
- **기간 열:** 세그먼트 또는 하위 세그먼트가 실행된 시간을 나열합니다.
- **상태 열:** 세그먼트 또는 하위 세그먼트 상태의 결과를 나열합니다.
- **마지막 열:** 타임라인의 다른 세그먼트 또는 하위 세그먼트와 비교하여 세그먼트 또는 하위 세그먼트가 실행된 기간에 해당하는 가로 막대를 표시합니다.

콘솔이 타임라인을 생성하는 데 사용하는 원시 트레이스 데이터를 보려면 원시 데이터 탭을 선택합니다. 원시 데이터에는 트레이스에 대한 정보와 트레이스를 구성하는 세그먼트와 하위 세그먼트에 대한 정보가 형식으로 표시됩니다. JSON 이 정보에는 다음이 포함될 수 있습니다.

- 타임스탬프
- 고유 ID
- 세그먼트 또는 하위 세그먼트와 관련된 리소스
- 세그먼트 또는 하위 세그먼트의 출처 또는 출처
- HTTP 요청의 응답과 같은 애플리케이션에 대한 요청에 대한 추가 정보.

계속된 AWS SDK 또는 SQL 클라이언트를 사용하여 외부 리소스를 호출하면 X-Ray SDK가 하위 세그먼트를 자동으로 기록합니다. HTTP 또한 X-Ray SDK를 사용하여 모든 함수 또는 코드 블록에 대한 사용자 지정 하위 세그먼트를 기록할 수 있습니다. 사용자 지정 하위 세그먼트가 열려 있는 동안 기록된 추가 하위 세그먼트는 사용자 지정 하위 세그먼트의 하위 세그먼트가 됩니다.

## 세그먼트 세부 정보 보기

추적 타임라인에서 세그먼트의 이름을 선택하여 세부 정보를 확인합니다.

세그먼트 세부 정보 패널에는 개요, 리소스, 주식, 메타데이터, 예외 및 SQL 탭이 표시됩니다. 다음이 적용됩니다.

- 개요 탭에 요청 및 응답에 대한 정보가 표시됩니다. 정보에는 이름, 시작 시간, 종료 시간, 기간, 요청 URL, 요청 작업, 요청 응답 코드, 오류 및 장애가 포함됩니다.
- 세그먼트의 리소스 탭에는 X-Ray-SDK의 정보와 애플리케이션을 실행하는 AWS 리소스에 대한 정보가 표시됩니다. X-Ray SDK용 Amazon EC2 또는 Amazon ECS 플러그인을 사용하여 서비스별 리소스 정보를 기록할 수 있습니다. AWS Elastic Beanstalk 플러그인에 대한 자세한 내용은 의 서비스 플러그인 섹션을 참조하십시오. [Java용 X-Ray SDK 구성](#)

- 나머지 탭에는 세그먼트에 기록된 주석, 메타데이터 및 예외가 표시됩니다. 계측된 요청에서 예외가 생성되면 자동으로 캡처됩니다. 주석과 메타데이터에는 X-Ray SDK가 제공하는 작업을 사용하여 기록하는 추가 정보가 포함됩니다. 세그먼트에 주석이나 메타데이터를 추가하려면 X-Ray SDK를 사용하세요. 자세한 내용은 SDK를 사용하여 애플리케이션 계측하기 아래에 나열된 언어별 링크를 참조하십시오. AWS X-Ray [애플리케이션 측정 대상 AWS X-Ray](#)

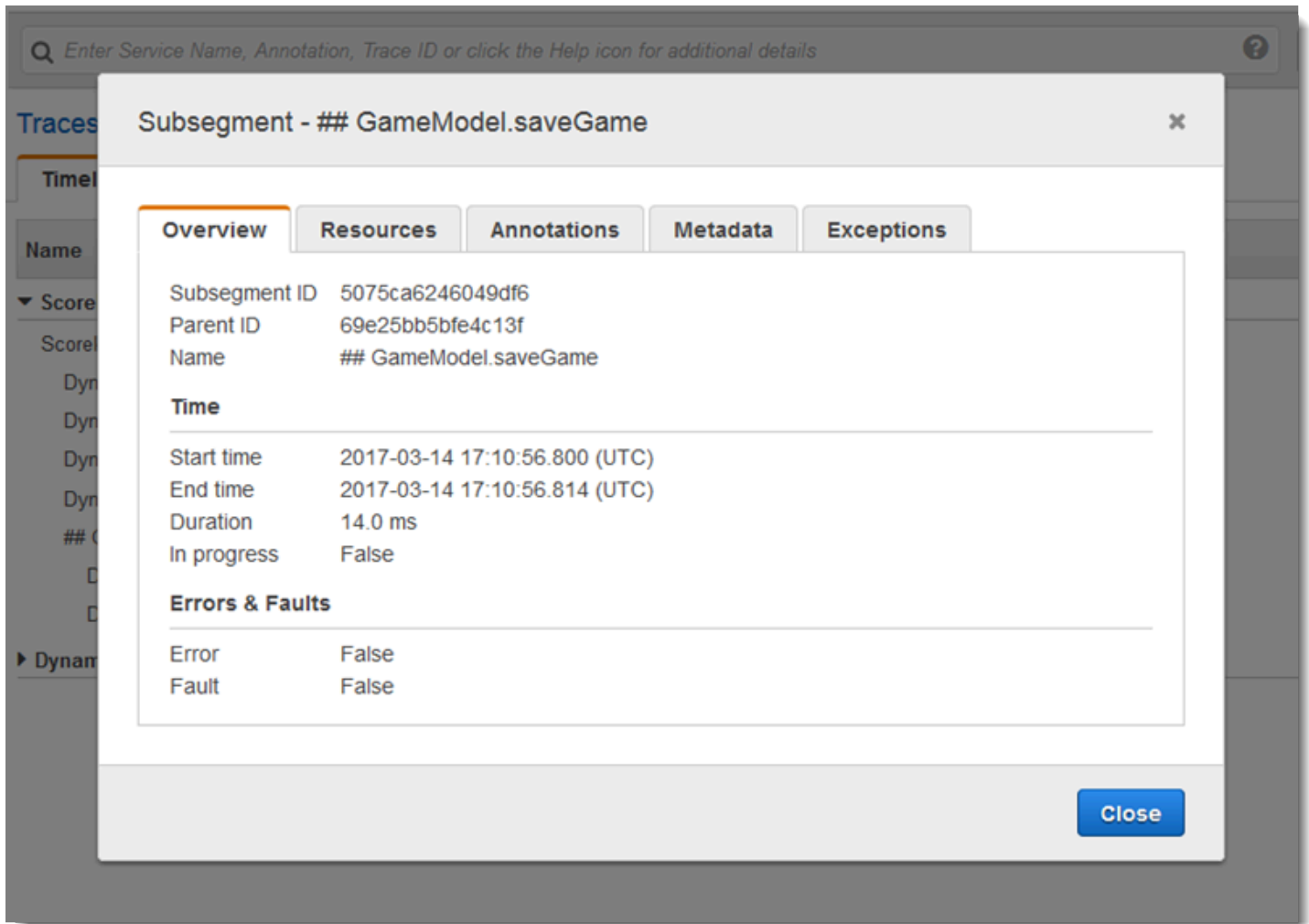
## 하위 세그먼트 세부 정보 보기

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, 요청URL, 요청 작업, 요청 응답 코드, 오류 및 오류가 포함됩니다. 구성된 클라이언트에 의해 생성된 하위 세그먼트의 경우, 개요 탭에 애플리케이션 관점에서의 요청 및 응답에 대한 정보가 포함됩니다.
- 하위 세그먼트의 리소스 탭에는 해당 하위 세그먼트를 실행하는 데 사용된 AWS 리소스에 대한 세부 정보가 표시됩니다. 예를 들어 리소스 탭에는 AWS Lambda 함수 ARN, DynamoDB 테이블에 대한 정보, 호출된 모든 작업, 요청 ID가 포함될 수 있습니다.
- 나머지 탭에는 하위 세그먼트에 기록된 주석, 메타데이터 및 예외가 표시됩니다. 계측된 요청에서 예외가 생성되면 자동으로 캡처됩니다. 주석과 메타데이터에는 X-Ray SDK가 제공하는 작업을 사용하여 기록하는 추가 정보가 포함됩니다. X-Ray SDK를 사용하여 세그먼트에 주석 또는 메타데이터를 추가할 수 있습니다. 자세한 내용은 SDK를 사용하여 애플리케이션 계측하기 아래에 나열된 언어별 링크를 참조하십시오. AWS X-Ray [애플리케이션 측정 대상 AWS X-Ray](#)

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 내용은 SDK를 사용하여 애플리케이션 인스트루먼트에 나열된 언어별 링크를 참조하십시오. AWS X-Ray [Java용 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, 오류 또는 장애가 포함됩니다.



사용자 지정 하위 세그먼트의 메타데이터 탭에는 해당 하위 세그먼트에서 사용하는 리소스에 대한 JSON 형식 정보가 들어 있습니다.

## 필터 표현식 사용

필터 표현식을 사용하여 특정 요청, 서비스, 두 서비스 간 연결 (에지) 또는 조건을 충족하는 요청에 대한 트레이스 맵 또는 트레이스를 볼 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.

## 필터 표현식 세부 정보

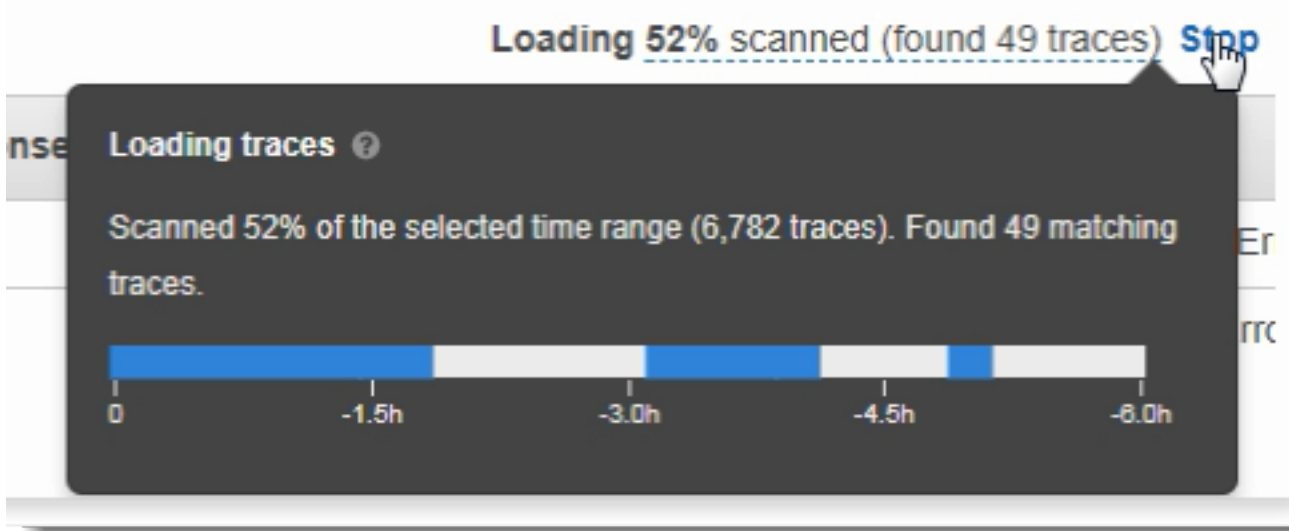
추적 맵에서 노드를 선택하면 콘솔은 노드의 서비스 이름과 선택에 따라 나타나는 오류 유형을 기반으로 필터 표현식을 구성합니다. 성능 문제를 표시하거나 특정 요청과 관련된 트레이스를 찾으려면 콘솔에서 제공하는 표현식을 조정하거나 자체적으로 표현식을 생성합니다. X-Ray SDK로 주석을 추가하는 경우 주석 키의 존재 또는 키 값을 기준으로 필터링할 수도 있습니다.

### Note

추적 맵에서 상대 시간 범위를 선택하고 노드를 선택하면 콘솔은 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.



그룹과 함께 필터 표현식을 사용하십시오.

그룹은 필터 표현식으로 정의한 추적 모음입니다. 그룹을 사용하여 추가 서비스 그래프를 생성하고 Amazon CloudWatch 지표를 제공할 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, 유니리 또는 바이너리 연산자, 비교를 위한 값을 포함할 수 있습니다.

*keyword operator value*

다양한 유형의 키워드에 대해 다양한 연산자를 사용할 수 있습니다. 예를 들어 responsetime은 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 시간이 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 찾으려면 복잡한 `service()` 키워드와 `edge()` 를 사용할 수 있습니다. 이러한 키워드를 사용하면 필터 표현식을 모든 다운스트림 노드, 단일 다운스트림 노드 또는 두 노드 간 에지에 적용할 수 있습니다. 이러한 키워드에 대한 자세한 내용은 다음 복합 키워드 섹션을 참조하십시오. 더욱 세분화하려면 `id()` 함수를 사용하여 서비스와 엣지를 필터링할 수 있습니다. 자세한 내용은 다음 `id` 함수 섹션을 참조하십시오.

### 부울 키워드

부울 키워드 값은 `true` 또는 `false`입니다. 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 `true`이면 표현식이 `true`입니다.
- `!` – 키워드가 `false`이면 표현식이 `true`입니다.
- `=, !=` – 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 결합 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.
- `edge(source, destination) {filter}` - `source` 서비스와 `destination` 서비스 사이를 연결합니다. 선택적으로, 이 연결의 세그먼트에 적용되는 필터 표현식을 종괄호로 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 항목을 생성하는 단계는 [AWS X-Ray에서 데이터 가져오기를](#) 참조하십시오.
- `service(name) {filter}` - 이름이 `name`인 서비스. 선택적으로, 서비스에서 생성하는 세그먼트에 적용되는 필터 표현식을 종괄호로 묶을 수 있습니다.

`service` 키워드를 사용하여 추적 맵의 특정 노드에 도달한 요청의 트레이스를 찾을 수 있습니다.

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 항목을 생성하는 단계는 [AWS X-Ray에서 데이터 가져오기를](#) 참조하십시오.

`service` 키워드를 사용하여 추적 맵의 특정 노드에 도달한 요청의 트레이스를 찾을 수 있습니다.

#### 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 장애를 일으킨 요청.

```
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.url이 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 gameid인 "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 오티지와 조합한 주석

```
service { annotation.request_id = "917DL6V0" }
```

```
edge { source.annotation.request_id = "916DL6V0" }
```

```
edge { destination.annotation.request_id = "918DL6V0" }
```

### 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 추적 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-name이라는 서비스에 대한 오류를 포함하는 요청

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

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

### Example – id 함수를 사용한 서비스 필터

유형이 random-name인 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::Function인 서비스의 오류를 포함하는 요청

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

특정 AWS 계정노드를 검색하려면 계정 ID를 지정하십시오.

### 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 계정 간 관찰 기능을 지원하므로 한 계정 내 여러 계정에 걸쳐 있는 애플리케이션을 모니터링하고 문제를 해결할 수 있습니다. AWS 리전연결된 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. [CloudWatch콘솔의](#) X-Ray trace 맵과 트레이스 페이지에서 계정 간 트레이스를 볼 수 있습니다.

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

- 아마존의 지표 CloudWatch
- Amazon CloudWatch 로그의 로그 그룹
- 트레이스 인 AWS X-Ray
- Amazon의 애플리케이션 CloudWatch 애플리케이션 인사이트

## 교차 계정 관찰성 구성

교차 계정 관찰성을 활성화하려면 하나 이상의 AWS 모니터링 계정을 설정하고 여러 소스 계정과 연결하세요. 모니터링 계정은 소스 계정에서 생성된 통합 가시성 데이터를 보고 상호 작용할 수 있는 AWS 계정인 중앙 계정입니다. 소스 계정은 포함된 리소스에 대한 가시성 데이터를 생성하는 AWS 계정 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. 자세한 내용은 [AWS X-Ray 요금 및 Amazon CloudWatch 가격을](#) 참조하십시오.

모니터링 계정과 소스 계정 간에 링크를 생성하려면 CloudWatch 콘솔 또는 AWS CLI 및 API의 새로운 Observability Access Manager 명령을 사용하십시오. 자세한 내용은 [CloudWatch 계정 간 옵저버빌리티](#)를 참조하십시오.

### Note

X-Ray 트레이스는 수령한 AWS 계정 곳으로 요금이 청구됩니다. [샘플링된](#) 요청이 둘 AWS 계정이상의 서비스에 걸쳐 있는 경우 각 계정은 별도의 트레이스를 기록하고 모든 트레이스는 동일한 트레이스 ID를 공유합니다. 교차 계정 옵저버빌리티 요금에 대한 자세한 내용은 [요금 및 Amazon AWS X-Ray 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. 추적 맵과 추적 페이지 모두에서 오른쪽 상단에 모니터링 계정 배지가 표시됩니다.

Monitoring account Last updated now

5m 15m 30m 1h 3h 6h Custom
↺ ⌵ ✂
Map view List view

### 트레이스 맵

CloudWatch 콘솔의 왼쪽 탐색 창의 X-Ray 트레이스에서 Trace Map을 선택합니다. 기본적으로 추적 맵에는 모니터링 계정으로 추적을 보내는 모든 소스 계정의 노드와 모니터링 계정 자체의 노드가 표시됩니다. 추적 맵의 왼쪽 상단에서 필터를 선택하여 계정 드롭다운을 사용하여 추적 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.

▼ TestLambda
View logs
View traces
Analyze traces
View dashboard

Lambda Function
Account: Monitoring account (1234567890123)

---

Metrics | Alerts | Response time distribution

추적 맵의 오른쪽 상단에서 목록 보기를 선택하여 서비스 노드 목록을 확인합니다. 서비스 노드 목록에는 모니터링 계정 및 구성된 모든 소스 계정의 서비스가 포함됩니다. 노드 필터에서 선택하여 계정 레이블 또는 계정 ID를 기준으로 노드 목록을 필터링합니다.

Alarms	Latency (avg)	Faults (5xx)
1	13ms	0.00/min

## 트레이스

모니터링 계정에서 CloudWatch 콘솔을 열고 왼쪽 탐색 창의 X-Ray 트레이스에서 트레이스를 선택하여 여러 계정에 걸친 트레이스의 트레이스 세부 정보를 확인합니다. X-Ray Trace Map에서 노드를 선택한 다음 노드 세부 정보 창에서 추적 보기를 선택하여 이 페이지를 열 수도 있습니다.

Traces(추적) 페이지는 계정 ID로 쿼리를 지원합니다. 시작하려면 하나 이상의 계정 ID가 포함된 쿼리를 입력합니다. 쿼리에 대한 자세한 내용은 [참조하십시오](#) [필터 표현식 사용](#). 다음 예시에서는 계정 ID X 또는 Y를 통과한 트레이스를 쿼리합니다.

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

Traces Info

Find traces by typing a query, build a query using the Query refiners section, or [choose a sample query](#). You can also [find a trace by ID](#).

Filter by X-Ray group

service(id(account.id: "1234567890123"))

Run query

5 traces retrieved

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

## ▼ Query refiners

Refine query by

Account

1 selected

Add to query

Select rows to filter traces

&lt; 1 &gt;



Account name and ID



Monitoring account (1234567890123)

## 트레이스 세부 정보

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

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

▼ TestLambda AWS::Lambda::Function Monitoring account (1234567890123)

TestLambda	✔ OK	-	28ms	
Invocation	✔ OK	-	1ms	
Overhead	✔ OK	-	8ms	

이벤트 기반 애플리케이션을 추적합니다.

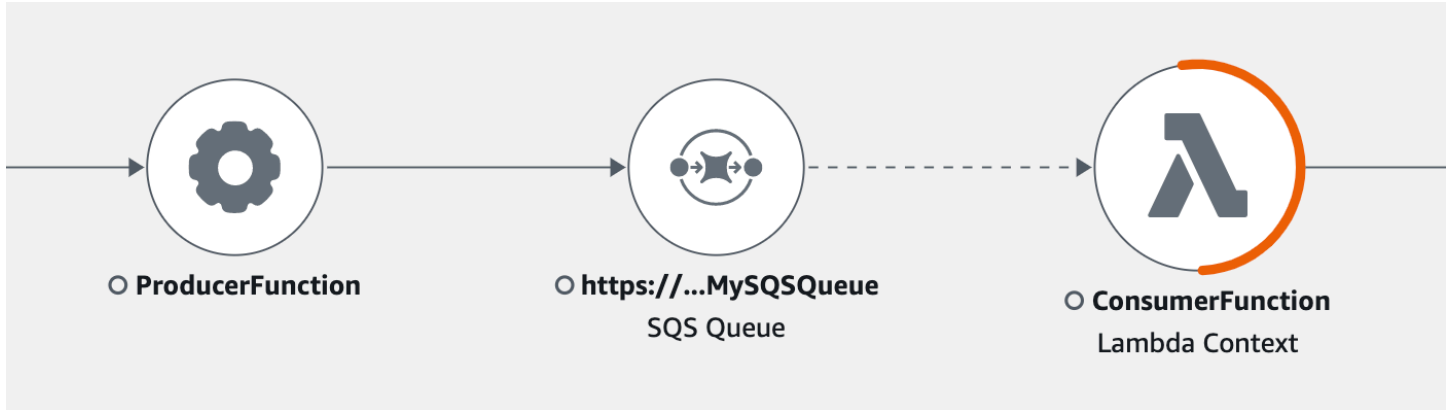
AWS X-Ray Amazon SQS 및 를 사용하는 이벤트 기반 애플리케이션 추적을 지원합니다. AWS Lambda CloudWatch 콘솔을 사용하면 Amazon SQS로 대기하고 하나 이상의 Lambda 함수로 처리되는 각 요청의 연결된 보기를 볼 수 있습니다. 업스트림 메시지 생산자의 트레이스는 다운스트림 Lambda 소비자 노드의 트레이스에 자동으로 연결되어 애플리케이션 뷰를 생성합니다. end-to-end

## Note

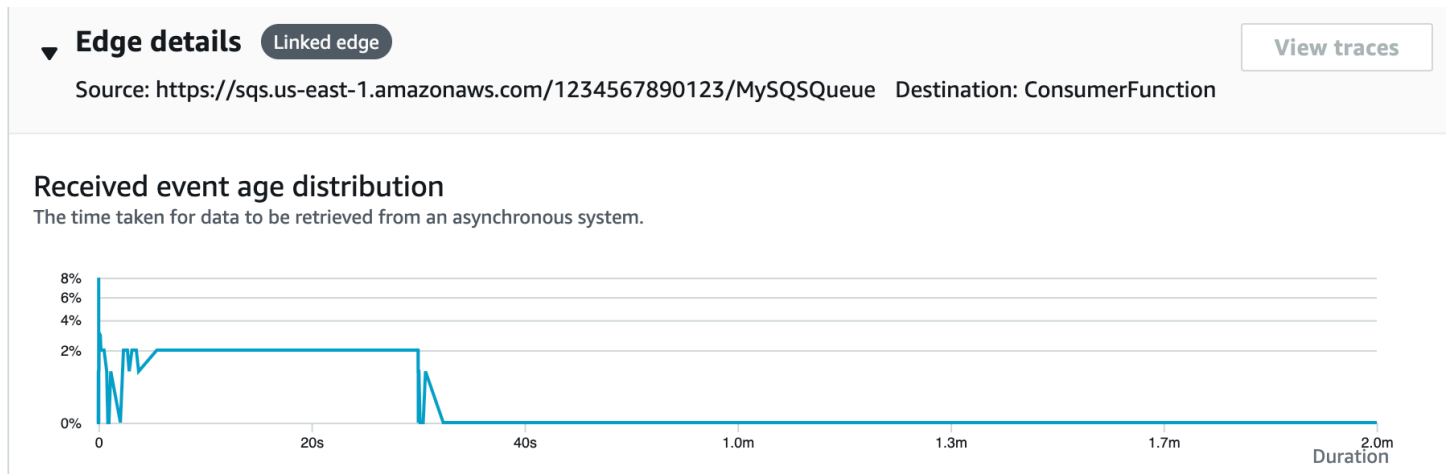
각 추적 세그먼트는 최대 20개의 추적에 연결할 수 있으며, 추적에는 최대 100개의 링크가 포함될 수 있습니다. 특정 시나리오에서 추가 추적을 연결하면 [최대 추적 문서 크기](#)를 초과하여 잠재적으로 불완전한 추적을 초래할 수 있습니다. 예를 들어 추적이 활성화된 Lambda 함수가 한 번의 간접 호출로 많은 SQS 메시지를 대기열에 전송하는 경우 이런 일이 발생할 수 있습니다. 이 문제가 발생하는 경우 X-Ray SDK를 사용하는 완화 방법을 사용할 수 있습니다. 자세한 내용은 [자바](#), [Node.js](#), [Python](#),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

## 추적 맵에서 연결된 추적 보기

[CloudWatch 콘솔](#) 내의 Trace Map 페이지를 사용하여 Lambda 소비자의 트레이스에 연결된 메시지 생산자의 트레이스가 포함된 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

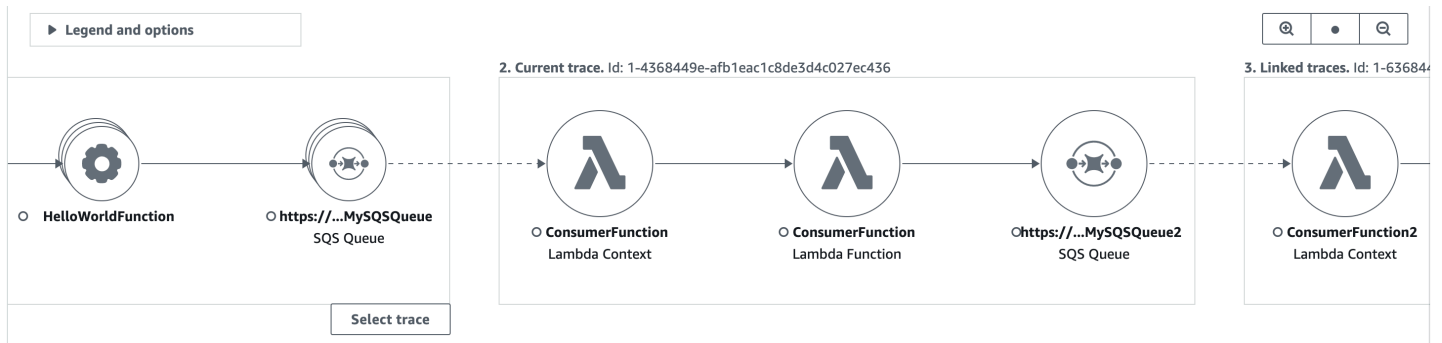
메시지 생산자, Amazon SQS 대기열 또는 람다 소비자로부터 전송된 추적 세부 정보를 확인합니다:

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

CloudWatch > Traces > Trace 1-4368449e-afb1eac1c8de3d4c027ec436

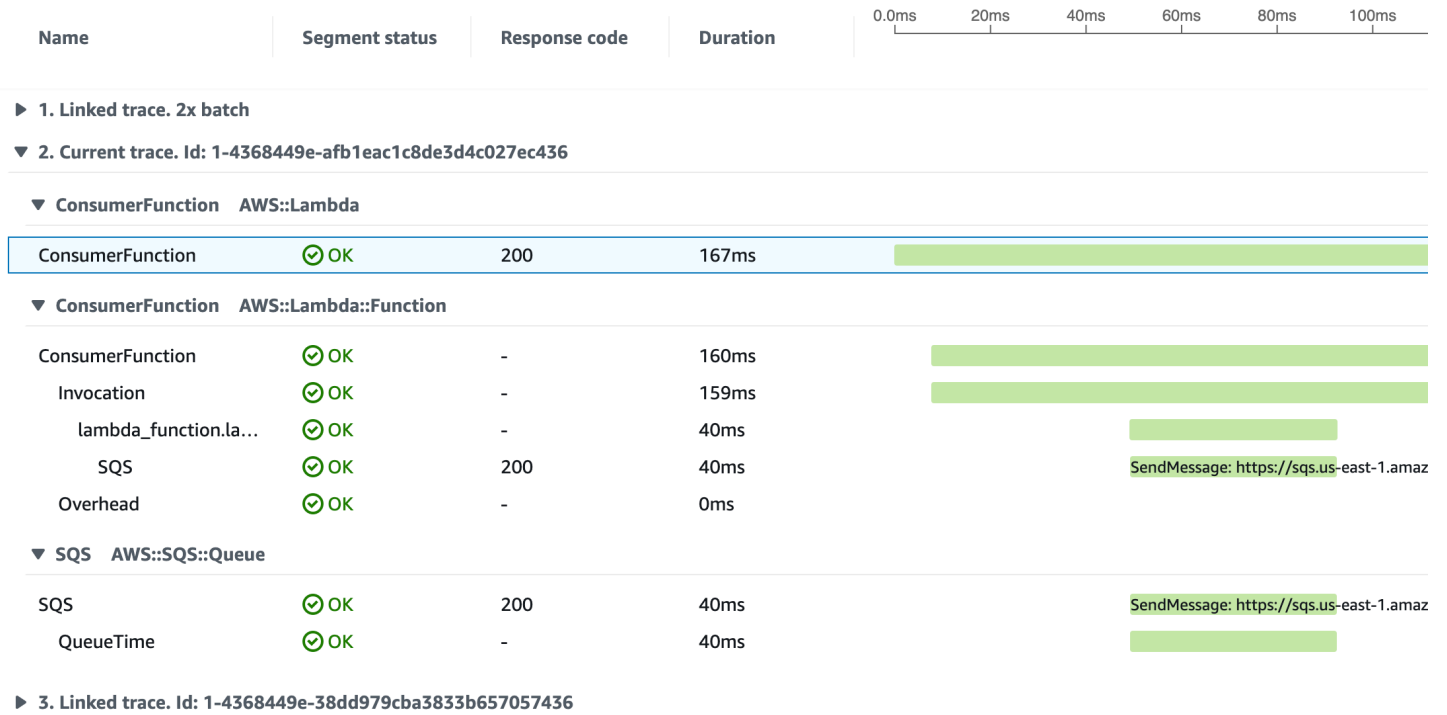
## Trace 1-6368449e-afb1eac1c8de3d4c027ec436 Info This trace is part of a linked set of traces

추적 세부 정보 맵에는 업스트림 및 다운스트림 연결 추적과 함께 현재 트레이스가 표시되며, 각 트레이스는 각 트레이스의 범위를 나타내는 상자 안에 들어 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 다운스트림 연결 트레이스를 포함한 트레이스 세그먼트의 타임라인이 표시됩니다. 업스트림 또는 다운스트림에 연결된 트레이스가 여러 개 있는 경우 해당 세그먼트 세부 정보를 표시할 수 없습니다. 연결된 트레이스 세트 내 단일 트레이스의 세그먼트 세부 정보를 보려면 다음 섹션에 설명된 대로 트레이스 하나를 선택하십시오.

Segments Timeline [Info](#)



연결된 트레이스 세트 내에서 단일 트레이스 선택하기

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

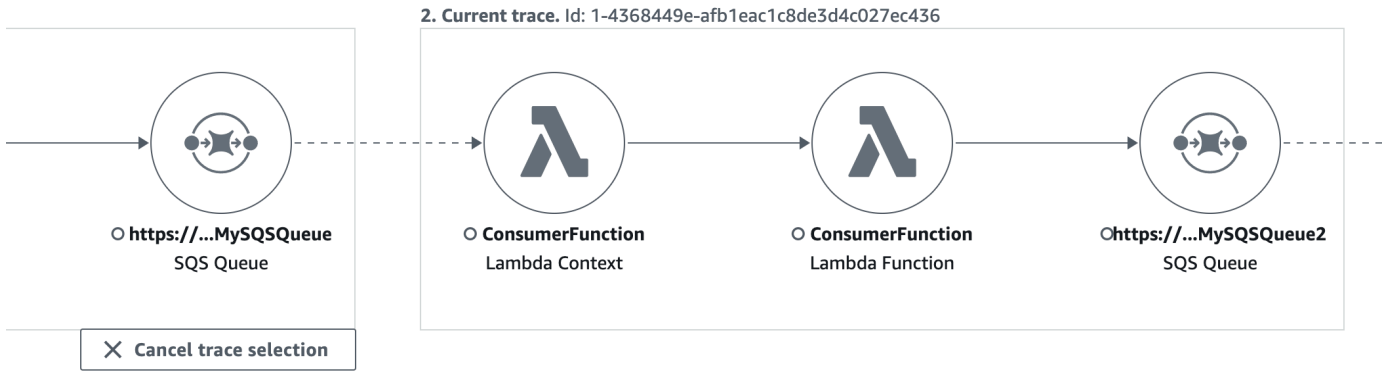
1. 트레이스 세부 정보 맵에서 연결된 트레이스 아래에 있는 트레이스 선택을 선택합니다. 추적 목록이 표시됩니다.

**Traces (2)**

	ID	Trace status	Timestamp	Response code
<input checked="" type="radio"/>	...3fd6e9600d58fea82597e9af	OK	11.7min (2022-11-06 15:34:54)	200
<input type="radio"/>	...223d41cc17bae4a5394423a0	OK	11.7min (2022-11-06 15:34:54)	200

2. 트레이스 옆의 라디오 버튼을 선택하면 트레이스 세부 정보 맵 내에서 트레이스를 볼 수 있습니다.
3. 연결된 트레이스의 전체 세트를 보려면 트레이스 선택 취소를 선택합니다.





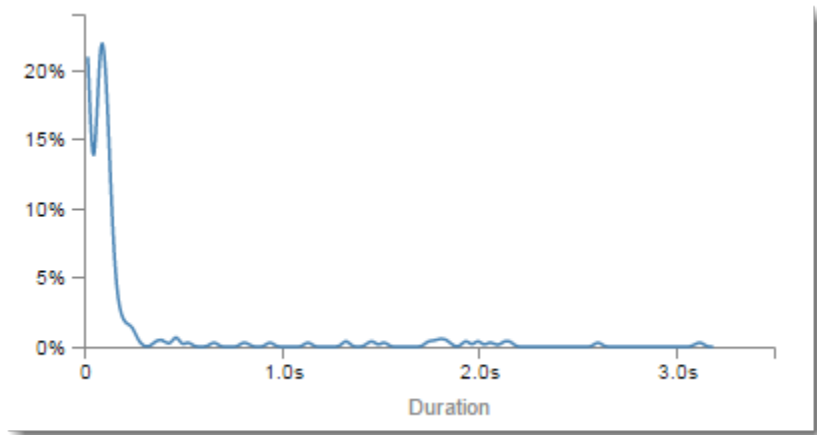
지연 시간 히스토그램을 사용하세요.

추적 맵에서 노드나 에지를 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.



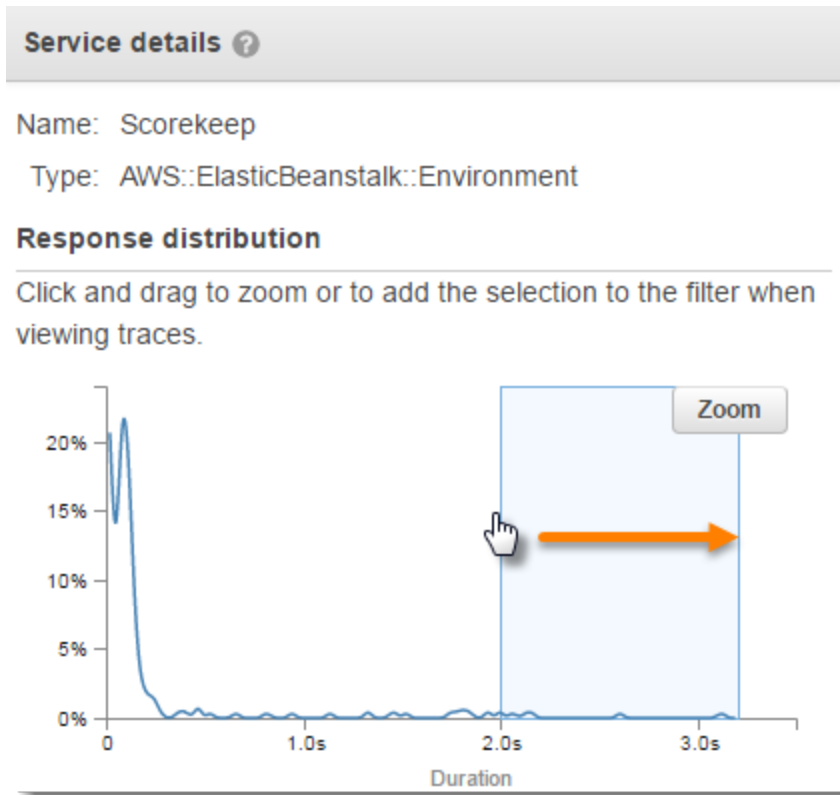
### 서비스 세부 정보 해석

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

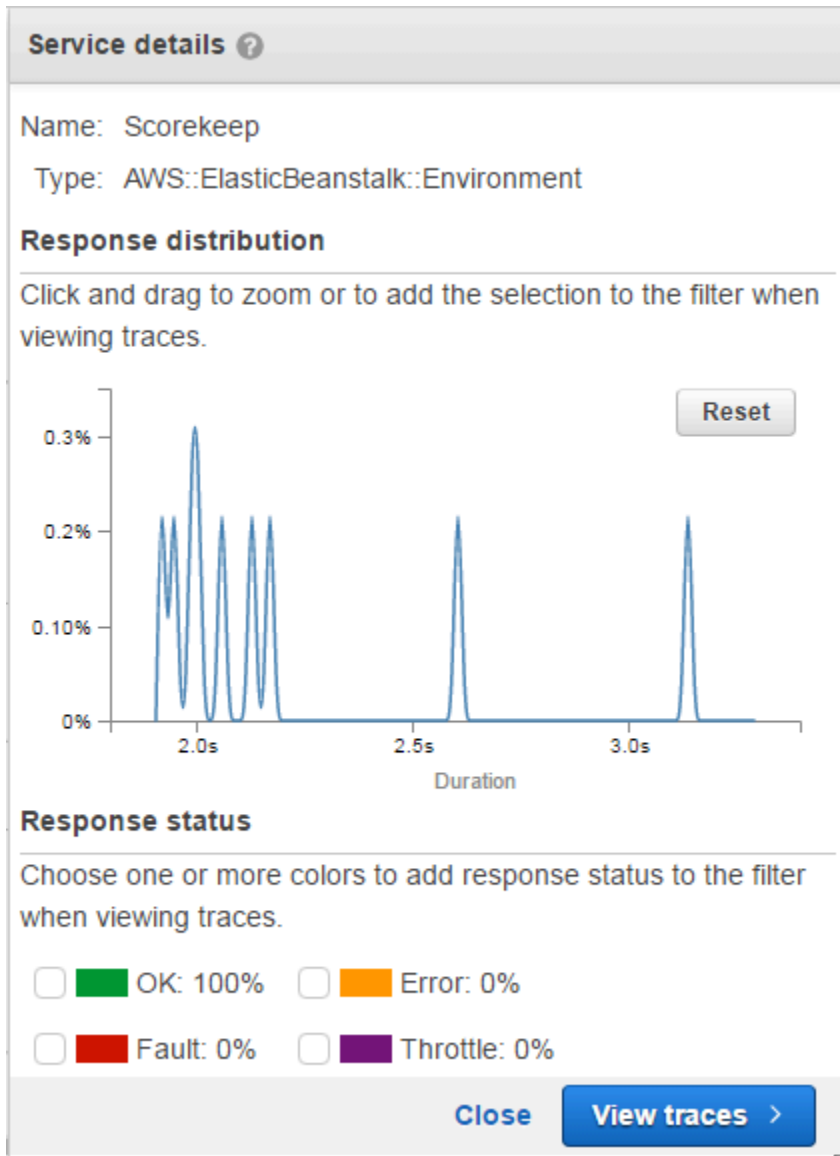
- 원을 클릭하여 서비스 노드를 선택합니다. X-Ray는 서비스에서 처리한 요청에 대한 히스토그램을 보여줍니다. 지연 시간은 서비스에 의해 기록된 것이며 서비스와 요청자 간 네트워크 지연 시간은 포함하지 않습니다.
- 두 서비스 간 가장자리의 선이나 화살표 끝을 클릭하여 엣지를 선택합니다. X-Ray는 다운스트림 서비스에서 처리한 요청자의 요청에 대한 히스토그램을 보여줍니다. 지연 시간은 요청자에 의해 기록된 것이며 두 서비스 간 네트워크 연결 지연 시간을 포함합니다.

[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 계정의 추적 데이터를 지속적으로 분석하여 애플리케이션에서 발생하는 긴급 문제를 식별합니다. 결함률이 예상 범위를 초과하면 문제를 기록하고 문제가 해결될 때까지 그 영향을 추적하는 인사이트를 생성합니다. Insights를 사용하면 다음을 수행할 수 있습니다.

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

- 시간이 지남에 따라 문제가 변경될 때마다 알림을 받습니다. EventBridgeAmazon을 사용하여 Insights 알림을 모니터링 및 경고 솔루션과 통합할 수 있습니다. 이 통합 기능을 사용하면 문제의 심각도에 따라 자동화된 이메일 또는 알림을 보낼 수 있습니다.

X-Ray 콘솔은 추적 맵에서 진행 중인 사고가 있는 노드를 식별합니다. 인사이트 요약을 보려면 해당 노드를 선택하십시오. 왼쪽 탐색 창에서 Insights를 선택하여 Insights를 보고 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. 위 예제에서 이상 현상은 의 결함 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 활성화

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

1. [X-Ray 콘솔](#)을 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 구성](#)를 참조하세요.
3. 왼쪽의 탐색 창에서 인사이트를 선택한 다음 확인하려는 인사이트를 선택합니다.

Description	Duration	Root cause service	Anomalous services	Group	Start time
Overall, 30% of the client requests failed due to faults and 19% of the requests to api (AWS::ElasticBeanstalk::Environment) failed due to faults. <span>Closed</span> <span>Fault</span>	2 minutes 58 seconds	api (AWS::ElasticBeanstalk::Envir...)	www (AWS::ElasticBeanstalk::Envir...) api (AWS::ElasticBeanstalk::Envir...)	Default	Jan 19th 2021, 19:02

### Note

X-Ray는 GetInsightSummaries GetInsight, GetInsightEvents, 및 GetInsightImpactGraph API 작업을 사용하여 인사이트로부터 데이터를 검색합니다. 통찰력을 보려면 AWSXrayReadOnlyAccess IAM 관리형 정책을 사용하거나 IAM 역할에 다음 사용자 지정 정책을 추가하십시오.

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

자세한 정보는 [AWS X-Ray IAM과의 작동 방식](#)을 참조하세요.

## 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 수신하

고 조건부 규칙을 사용하여 SNS 알림, Lambda 호출, SQS 대기열에 메시지 게시 또는 대상 지원 등의 조치를 취할 수 있습니다. EventBridge 인사이트 알림은 최선의 노력을 다해 발송되지만 반드시 발송된다는 보장은 없습니다. 대상에 대한 자세한 내용은 [Amazon EventBridge 대상을](#) 참조하십시오.

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

X-Ray 그룹에 대한 알림을 활성화하려면

1. [X-Ray 콘솔](#)을 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 구성](#)를 참조하세요.

Amazon EventBridge 조건부 규칙을 구성하려면

1. [Amazon EventBridge 콘솔](#)을 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" ]
  }
}
```

- 이벤트가 이 규칙과 일치할 때 간접 호출할 대상을 선택하고 구성합니다.
- (선택 사항) 이 규칙을 더 쉽게 식별하고 선택할 수 있도록 태그를 입력합니다.
- 생성을 선택합니다.

#### Note

X-Ray Insights 알림은 현재 고객 관리 키를 지원하지 않는 EventBridge Amazon으로 이벤트를 전송합니다. 자세한 정보는 [AWS X-Ray의 데이터 보호](#)을 참조하세요.

## 인사이트 개요

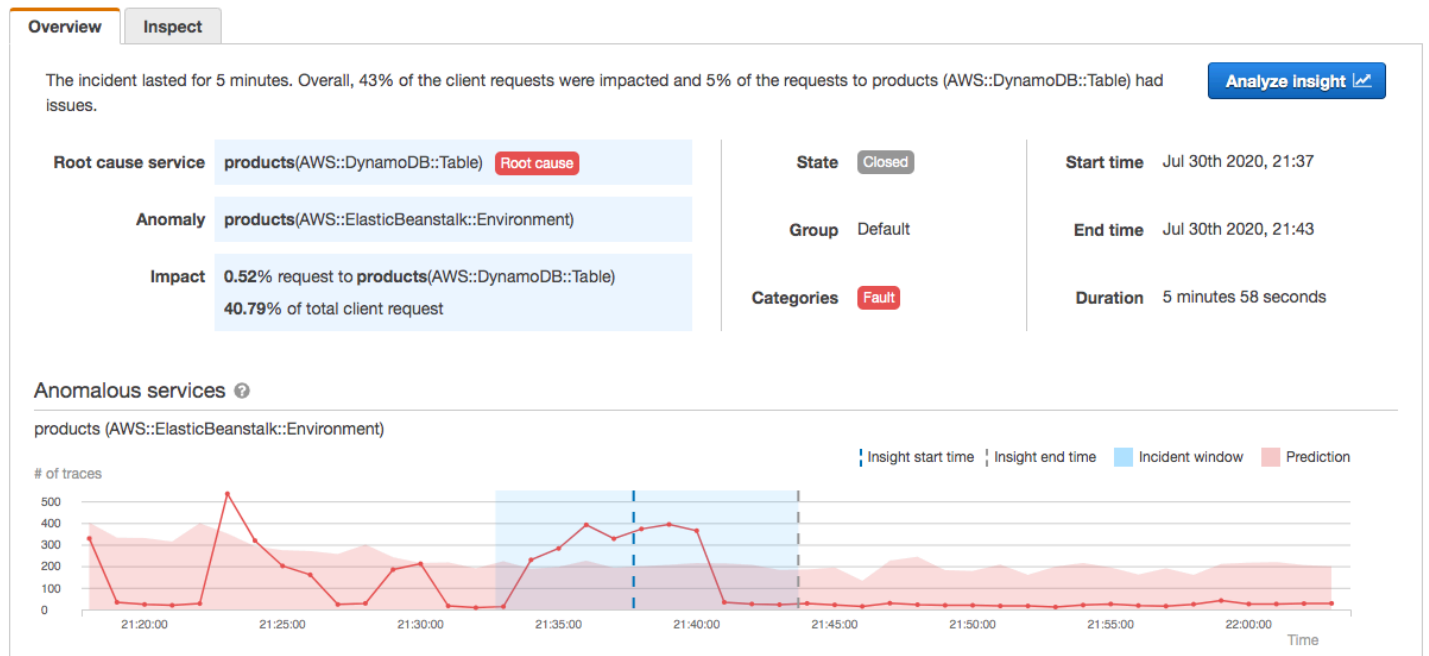
인사이트 개요 페이지에서는 세 가지 주요 질문에 대한 답변을 제시합니다:

- 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:

products (AWS::DynamoDB::Table) of Default group



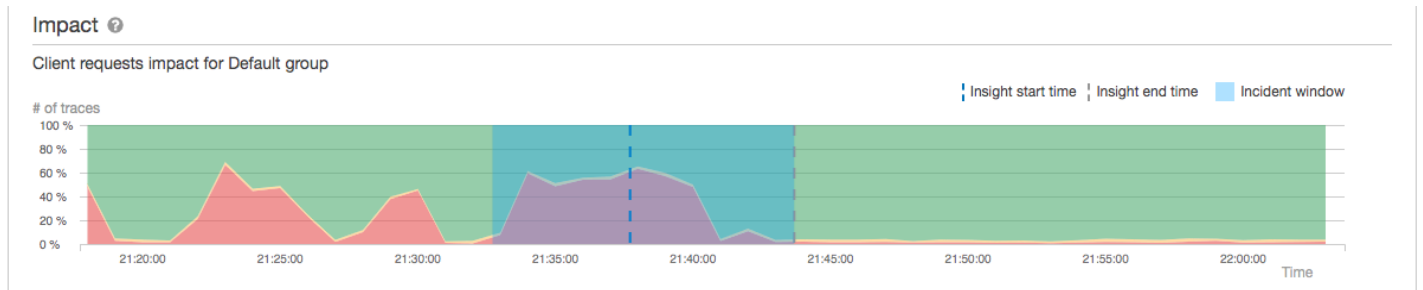
근본 원인 섹션에는 근본 원인 서비스 및 영향을 받는 경로에 초점을 맞춘 추적 맵이 표시됩니다. 근본 원인 맵의 오른쪽 상단에 있는 눈 아이콘을 선택하면 영향을 받지 않은 노드를 숨길 수 있습니다. 근본 원인 서비스는 X-Ray가 이상 징후를 식별한 가장 멀리 위치한 다운스트림 노드입니다. 이는 사용자가 계측한 서비스 또는 서비스가 계측된 클라이언트로 호출한 외부 서비스를 나타낼 수 있습니다. 예를 들어 AWS 계측된 SDK 클라이언트를 사용하여 Amazon DynamoDB를 호출하는 경우 DynamoDB의 장애가 증가하면 DynamoDB를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

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 정보는 [애널리틱스 콘솔과 상호 작용하세요](#)를 참조하세요.





맵의 업스트림에서 계속해서 발생하는 오류는 여러 노드에 영향을 미치고 여러 이상 징후를 유발할 수 있습니다. 오류가 요청을 한 사용자에게까지 전달되면 클라이언트 오류가 발생하게 됩니다. 이는 추적 맵의 루트 노드에 발생한 결함입니다. 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 정보는 [애널리틱스 콘솔과 상호 작용하세요](#)를 참조하세요.

## 영향력 이해하기

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

- X-Ray 그룹에 미치는 영향. 자세한 내용은 [그룹 구성](#)을 참조하십시오.
- 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 마이크로서비스 전반의 중복 문제를 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, 각 주목할 만한 중간 변경 사항을 알림으로 기록하여 Amazon 이벤트로 전송할 수 있습니다. EventBridge 이를 통해 프로세스와 워크플로를 구

축하여 시간이 지남에 따라 문제가 어떻게 변했는지 파악하고 이메일을 보내거나 를 사용하여 경고 시스템과 통합하는 등의 적절한 조치를 취할 수 있습니다. EventBridge

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

products (AWS::DynamoDB::Table) of Default group

**Overview** **Inspect**

You can use this section to investigate the progress of the insight by choosing an event on the timeline, and then viewing the impact and the corresponding incident graph.

**Impact timeline** (5 Events)

- Jul 30th 2020, 21:43 (25 days ago)
  - 40.66% Requests to group
  - 8.27% Impact decrease
- Insight is closed
- Jul 30th 2020, 21:42 (25 days ago)**
  - 48.93% Requests to group
  - 9.44% Impact decrease
  - 0.72% Requests to service
  - 0.15% Impact decrease
  - products Most impacted
- Jul 30th 2020, 21:39 (25 days ago)
  - 58.37% Requests to group
  - 11.32% Impact increase
  - 0.87% Requests to service
  - 0.29% Impact increase
  - products Most impacted

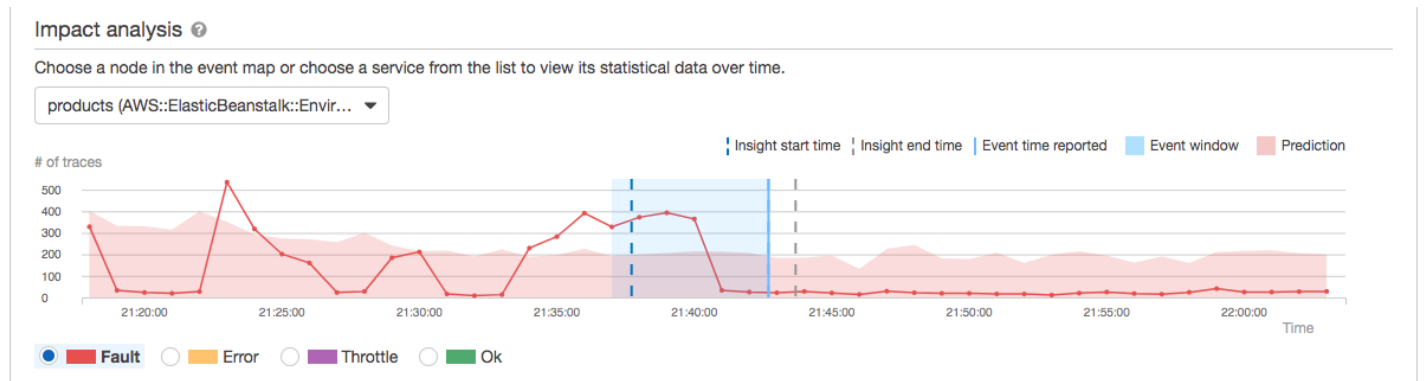
**Details for Jul 30th 2020, 21:42**

6% of the requests to products are now having issues.  
 Client impact has decreased; 49% of the client requests are now impacted.  
 Since the start of the incident, 43% of the client requests were impacted and 5% of the requests to products (AWS::DynamoDB::Table) had issues.

Map time range : 2020-07-30T21:37:00~2020-07-30T21:42:00

Trace map showing Client, API (AWS::ElasticBeanstalk::Environment), and products (AWS::DynamoDB::Table) with anomalies and a root cause identified.

이벤트에 대한 추적 맵과 그래프를 보려면 영향 타임라인에서 이벤트를 선택하십시오. 트레이스 맵은 인시던트의 영향을 받는 애플리케이션 내 서비스를 보여줍니다. 영향 분석은 선택한 노드와 그룹 내 클라이언트의 장애 타임라인을 그래프로 표시합니다.



인시던트와 관련된 트레이스를 더 자세히 살펴보고 싶다면 검사 페이지에서 이벤트 분석을 선택하십시오. 분석 페이지를 사용하여 추적 목록을 세분화하고 영향을 받는 사용자를 식별할 수 있습니다. 자세한 정보는 [애널리틱스 콘솔과 상호 작용하세요](#)를 참조하세요.

애널리틱스 콘솔과 상호 작용하세요.

AWS X-Ray Analytics 콘솔은 추적 데이터를 해석하여 애플리케이션과 기본 서비스가 어떻게 작동하는지 빠르게 이해할 수 있는 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.

## 콘솔 기능

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

## 특성

기능	설명
그룹	처음에 선택되는 그룹은 Default입니다. 검색된 그룹을 변경하려면 메인 필터 표현식 검색창 오른쪽에 있는 메뉴에서 다른 그룹을 선택하십시오. 그룹에 대해 자세히 알아보려면 <a href="#">그룹 구성</a> 을 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.

기능	설명
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 Filterd 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. 검색된 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 Filterd trace set B(필터링된 추적 집합 B) 타일 내에서 태그로 채워지며, 타일에서 필터를 제거할 수도 있습니다.

기능	설명
Response Time Root Cause Entity Paths(응답 시간 근본 원인 개체 경로)	기록된 개체 경로의 테이블입니다. X-Ray는 트race에서 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 의 근본 원인 분석 검색 및 구체화 섹션을 참조하십시오. <a href="#">X-Ray에서 데이터 가져오기</a>
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 트race를 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).

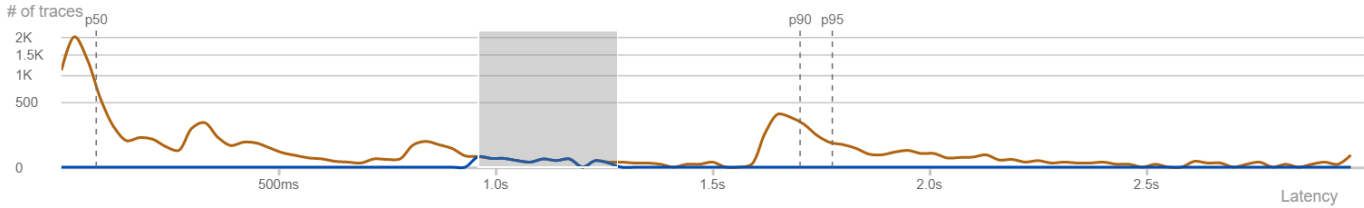
- All traces in the group(그룹 내 모든 트race – 회색)
- Retrieved traces(검색된 트race) – 주황색
- Filtered trace set A(필터링된 트race 집합 A) – 녹색
- Filtered trace set B(필터링된 트race 집합 B) – 파란색

### Example – 응답 시간 분포

응답 시간 분포란 임의 응답 시간을 갖는 트race 수를 나타낸 차트를 말합니다. 응답 시간 분포 내에서 선택하려면 클릭하여 끌어오면 됩니다. 그러면 특정 응답 시간에 속하는 모든 트race를 대상으로 responseTime이라고 하는 유효 트race 집합에 필터가 선택 및 생성됩니다.

## Response time distribution

Click and drag to filter the traces by response time.



## 시계열 활동

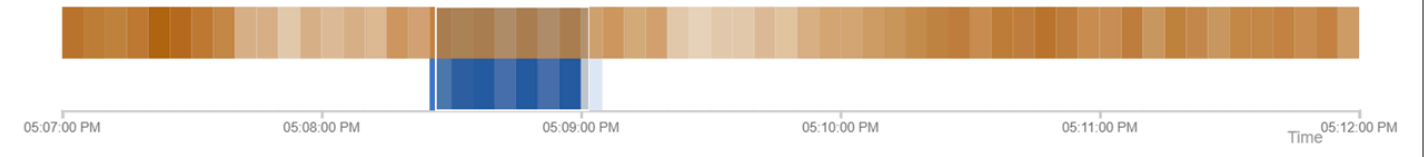
시계열 활동 차트에는 특정 시간의 범위에 속하는 트레이스 수가 표시됩니다. 색상 표시기에는 응답 시간 분포의 선 그래프 색상이 그대로 반영됩니다. 연속된 활동 내에서 색상 블록이 더욱 어둡고 진할수록 해당 시간에 트레이스가 더욱 많다는 것을 의미합니다.

## 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.

## Time series activity

Click and drag to filter the traces by time.



## 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

추적 맵은 오류 및 장애에 대한 성공적인 호출의 비율을 기준으로 색상을 지정하여 각 노드의 상태를 나타냅니다. 노드에서 빨간색 비율은 결함을 의미합니다. 이때는 X-Ray 분석 콘솔을 사용해 조사하십시오.

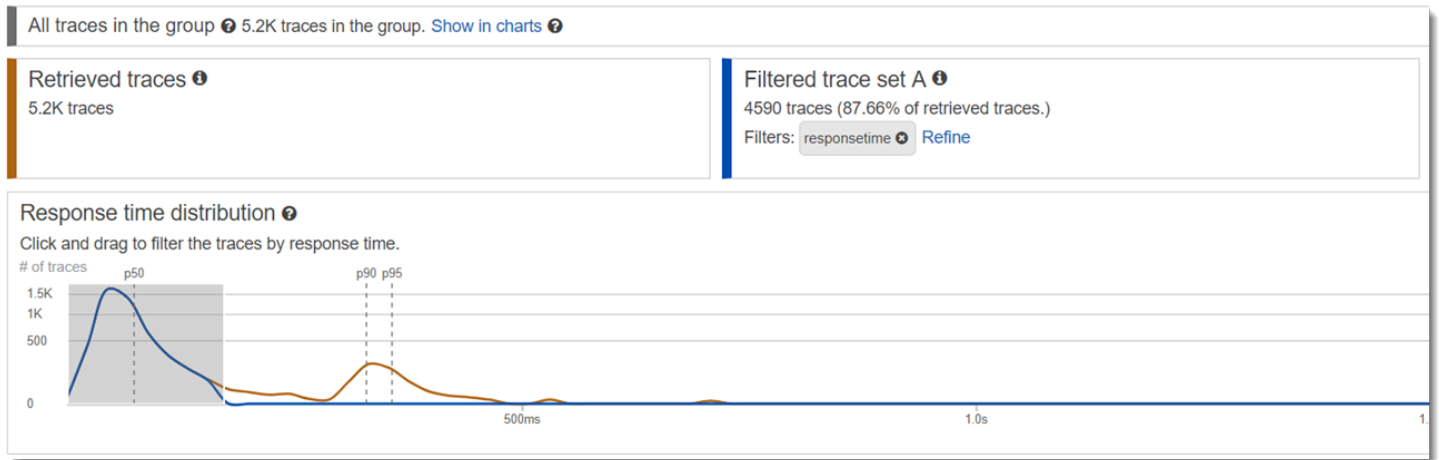
추적 맵을 읽는 방법에 대한 자세한 내용은 [X-Ray 추적 맵 사용](#)을 참조하십시오.



## 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

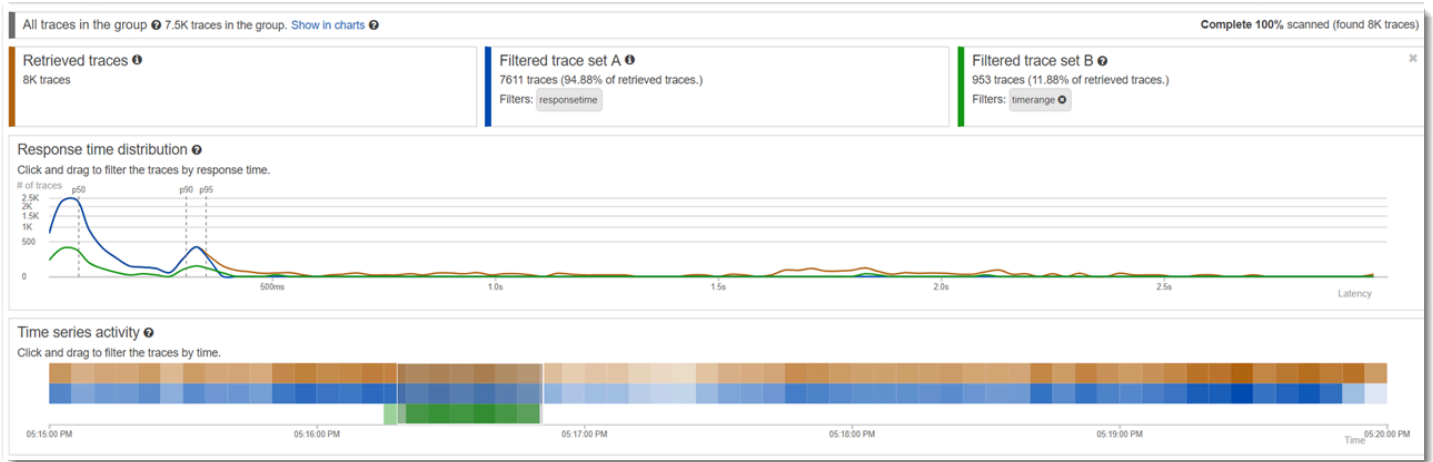
사용자, URL, 응답 시간 근본 원인 또는 기타 사전 설정된 속성을 기준으로 오류 집합을 자세하게 살펴봅니다. 예를 들어 상태 코드가 500인 트레이스 집합을 추가로 필터링한다고 가정할 경우, 먼저 USERS 테이블에서 행을 선택합니다. 그러면 트레이스 집합 타일에 앞서 지정했던 `http.status`와 `user`까지 필터 태그 2개가 생성됩니다.

## 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.





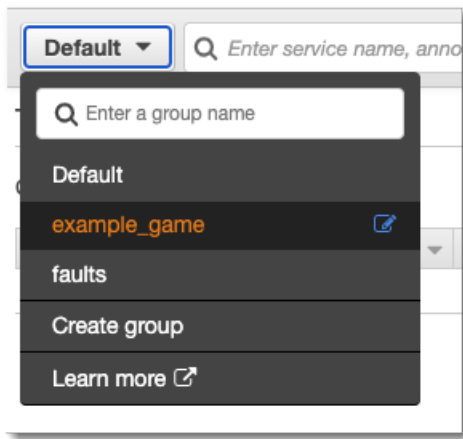
## 원하는 트레이스 식별 및 세부 정보 보기

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 생성하고 Amazon CloudWatch 지표를 제공할 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 [X-Ray API를 사용하여 샘플링, 그룹 및 암호화 설정 구성](#)을 참조하십시오.

트레이스 맵, 트레이스 또는 분석을 위한 트레이스 그룹을 생성할 수 있습니다. 그룹을 생성하면 트레이스 맵, 트레이스, 애널리틱스의 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 사용](#)을 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 요금](#)을 참조하십시오.

## 그룹 생성

#### Note

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

## CloudWatch console

1. <https://console.aws.amazon.com/cloudwatch/> 에서 AWS Management Console 로그인하고 CloudWatch 콘솔을 엽니다.

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 사용](#)을 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.

Enable insights

Enable notifications

Deliver insight events using Amazon EventBridge.

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](#) 단원을 참조하세요.

Key	Value - optional
<input type="text" value="Q Enter key"/>	<input type="text" value="Q Enter value"/>
<input type="button" value="Remove"/>	

9. 그룹 생성을 선택합니다.

## X-Ray console

1. <https://console.aws.amazon.com/xray/home> 에서 AWS Management Console 로그인하고 X-Ray 콘솔을 엽니다.
2. 왼쪽 탐색 창의 그룹 페이지 또는 추적 맵, 추적 및 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 사용](#)을 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.

Enable Insights

Enable Notifications  Deliver insight events using Amazon EventBridge. Learn more about Data Protection in EventBridge. [Learn more](#) 

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](#) 단원을 참조하세요.

application	game	
stage	prod	
Key	Value (optional)	

7. 그룹 생성을 선택합니다.

## 그룹 적용

### CloudWatch console

1. AWS Management Console 로그인하고 <https://console.aws.amazon.com/cloudwatch/>에서 CloudWatch 콘솔을 엽니다.
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. <https://console.aws.amazon.com/xray/home> 에서 AWS Management Console 로그인하고 X-Ray 콘솔을 엽니다.
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 AWS Management Console 로그인하고 <https://console.aws.amazon.com/cloudwatch/> 에서 CloudWatch 콘솔을 엽니다.
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 사용](#)을 참조하십시오. 다음 예제에서 그룹은 요청 URL 주소에 example/game가 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.

Enable insights

Enable notifications

Deliver insight events using Amazon EventBridge.

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](#) 단원을 참조하세요.

Key	Value - optional
<input type="text" value="Q Enter key"/>	<input type="text" value="Q Enter value"/>
<input type="button" value="Remove"/>	

8. 그룹 업데이트를 마치면 그룹 업데이트를 선택합니다.

## X-Ray console

1. <https://console.aws.amazon.com/xray/home> 에서 AWS Management Console 로그인하고 X-Ray 콘솔을 엽니다.
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 사용](#)을 참조하십시오. 다음 예제에서 그룹은 요청 URL 주소에 example/game가 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.

Enable Insights

Enable Notifications  Deliver insight events using Amazon EventBridge. Learn more about Data Protection in EventBridge. [Learn more](#)

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](#) 단원을 참조하세요.

application	game	X
stage	prod	X
Key	Value (optional)	X

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 -clone이 추가됩니다.

## CloudWatch console

1. 에 AWS Management Console 로그인하고 <https://console.aws.amazon.com/cloudwatch/> 에서 CloudWatch 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 설정을 선택합니다.
3. X-Ray 추적 섹션의 그룹에서 설정 보기를 선택합니다.
4. 그룹 섹션에서 그룹을 선택한 다음 복제를 선택합니다.
5. 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.
6. 기존 그룹의 필터 표현식을 그대로 유지하거나 선택적으로 새 필터 표현식을 입력할 수 있습니다. 필터 표현식을 작성하는 방법에 대한 자세한 내용은 [필터 표현식 사용](#)을 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 }
```

- 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](#) 단원을 참조하세요.
- 그룹 생성을 선택합니다.

## X-Ray console

- <https://console.aws.amazon.com/xray/home> 에서 AWS Management Console 로그인하고 X-Ray 콘솔을 엽니다.
- 왼쪽 탐색 창에서 그룹 페이지를 열고 복제할 그룹 이름을 선택합니다.
- [작업(Actions)] 메뉴에서 [그룹 복제(Clone group)]를 선택합니다.
- 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.
- 기존 그룹의 필터 표현식을 그대로 유지하거나 선택적으로 새 필터 표현식을 입력할 수 있습니다. 필터 표현식을 작성하는 방법에 대한 자세한 내용은 [필터 표현식 사용](#)을 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 }
```

- 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](#) 단원을 참조하세요.
- 그룹 생성을 선택합니다.

## 그룹 삭제

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



## CloudWatch console

1. <https://console.aws.amazon.com/cloudwatch/> 에서 AWS Management Console 로그인하고 CloudWatch 콘솔을 엽니다.
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. <https://console.aws.amazon.com/xray/home> 에서 AWS Management Console 로그인하고 X-Ray 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 그룹 페이지를 열고 삭제할 그룹 이름을 선택합니다.
3. 작업 메뉴에서 그룹 삭제를 선택합니다.
4. 확인 메시지가 표시되면 삭제를 선택합니다.

## Amazon에서 그룹 지표 보기 CloudWatch

그룹이 생성된 후 수신 추적은 X-Ray 서비스에 저장될 때 그룹의 필터 표현식에 대해 확인됩니다. 각 기준과 일치하는 트레이스 수에 대한 지표는 CloudWatch 1분마다 Amazon에 게시됩니다. 그룹 편집 페이지에서 지표 보기를 선택하면 CloudWatch 콘솔에서 지표 페이지가 열립니다. CloudWatch 메트릭을 사용하는 방법에 대한 자세한 내용은 Amazon 사용 CloudWatch 설명서의 [Amazon CloudWatch 메트릭스](#) 사용을 참조하십시오.

## CloudWatch console

1. AWS Management Console 로그인하고 <https://console.aws.amazon.com/cloudwatch/> 에서 CloudWatch 콘솔을 엽니다.
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 보기를 선택합니다.

CloudWatch 콘솔 메트릭 페이지가 새 탭에 열립니다.

## X-Ray console

1. <https://console.aws.amazon.com/xray/home> 에서 AWS Management Console 로그인하고 X-Ray 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 그룹 페이지를 열고 지표를 보려는 그룹의 이름을 선택합니다.
3. 그룹 편집 페이지에서 지표 보기를 선택합니다.

CloudWatch 콘솔 메트릭 페이지가 새 탭에 열립니다.

샘플링 규칙을 구성합니다.

AWS X-Ray 콘솔을 사용하여 서비스의 샘플링 규칙을 구성할 수 있습니다. 샘플링 구성을 통한 [능동적 추적을 AWS 서비스](#) 지원하는 X-Ray SDK는 샘플링 규칙을 사용하여 기록할 요청을 결정합니다.

샘플링 규칙을 구성합니다.

다음 사용 사례에 대한 샘플링을 구성할 수 있습니다.

- API 게이트웨이 진입점 — API 게이트웨이는 샘플링 및 활성 추적을 지원합니다. API 단계에서 적극 추적을 활성화하려면 [Amazon API Gateway 액티브 트레이싱 지원 대상 AWS X-Ray](#) 단원을 참조하십시오.
- AWS AppSync — 샘플링 및 활성 추적을 AWS AppSync 지원합니다. AWS AppSync 요청에 대한 활성 추적을 활성화하려면 [AWS X-Ray를 사용한 추적을](#) 참조하십시오.
- 컴퓨팅 플랫폼 기반 인스트루먼트 X-Ray SDK — Amazon EC2, Amazon ECS AWS Elastic Beanstalk 또는 같은 컴퓨팅 플랫폼을 사용하는 경우 애플리케이션이 최신 X-Ray SDK로 계측된 경우 샘플링이 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 기존 세트에 대해 얼마나 많은 요청을 기록할지 알려줍니다. 기본적으로 X-Ray SDK는 1초가 시작될 때 수신된 첫 번째 요청과 모든 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 샘플링

되는 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. AWS SDK에 대한 자세한 내용은 SDK [사용을](#) 참조하십시오.

### 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. <https://console.aws.amazon.com/cloudwatch/> 에서 AWS Management Console 로그인하고 CloudWatch 콘솔을 엽니다.
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 콘솔](#)을 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) - 추적 맵에 나타나는 계측 서비스의 이름입니다.
  - X-Ray SDK - 레코더에서 구성한 서비스 이름.
  - Amazon API Gateway - *api-name/stage*.
- 서비스 유형 (문자열) - 추적 맵에 나타나는 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 요청.
- 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 서비스를 구성합니다.

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 [Amazon API Gateway 액티브 트레이싱 지원 대상 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 [X-Ray API를 사용하여 샘플링, 그룹 및 암호화 설정 구성](#) 단원을 참조하십시오.

X-Ray SDK와 X-Ray API를 사용하여 샘플링 규칙을 읽고, 샘플링 결과를 보고하고, 샘플링 대상을 얻을 AWS 서비스 수도 있습니다. 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 API를 샘플링에 사용하는 방법에 대한 자세한 내용은 [X-Ray API에 샘플링 규칙 사용](#)을 참조하십시오.

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

## 콘솔 심층 연결

경로와 쿼리를 사용하여 특정 트레이스에 딥링크하거나 트레이스 및 트레이스 맵의 필터링된 보기를 만들 수 있습니다.

## 콘솔 페이지

- 시작 페이지 – [xray/home#/welcome](#)
- 시작하기 – [xray/home#/getting-started](#)

- 트레이스 맵 — [xray/home#/서비스 맵](#)
- 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 6시간 까지 가능합니다.

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



## Example — 지난 1시간 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
```

## 지역

해당 지역의 페이지로 AWS 리전 연결하려면 `an`을 지정합니다. 리전을 지정하지 않을 경우 콘솔이 마지막으로 방문한 리전으로 리디렉션됩니다.

리전 – `xray/home?region=region#/page`

## Example — 미국 서부 (오레곤) 의 트레이스 맵 (us-west-2)

```
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 (오레곤) 의 지난 시간 동안의 트레이스 맵 (us-west-2)

```
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

## SDK 사용

명령줄 인터페이스를 사용하거나 에서 사용할 수 있는 것보다 더 많은 사용자 지정 추적, 모니터링 또는 로깅 기능이 필요한 경우 SDK를 사용하세요. AWS Management Console AWS SDK를 사용하여 X-Ray API를 사용하는 프로그램을 개발할 수도 있습니다. AWS 배포판 OpenTelemetry (ADOT) SDK 또는 X-Ray SDK를 사용할 수 있습니다.

SDK를 사용하는 경우 애플리케이션을 계측할 때와 컬렉터 또는 에이전트를 구성할 때 워크플로에 사용자 지정을 추가할 수 있습니다. SDK를 사용하여 수행할 수 없는 다음과 같은 작업을 수행할 수 있습니다. AWS Management Console

- 사용자 지정 지표 게시 — 최대 1초의 고해상도로 지표를 샘플링하고, 여러 측정기준을 사용하여 측정항목에 대한 정보를 추가하고, 데이터 포인트를 통계 집합으로 집계합니다.
- 컬렉터 사용자 지정 — 수신기, 프로세서, 익스포터, 커넥터 등 컬렉터의 모든 부분에 맞게 구성을 사용자 지정할 수 있습니다.
- 계측 사용자 지정 — 세그먼트와 하위 세그먼트를 사용자 지정하고, 사용자 지정 키-값 쌍을 속성으로 추가하고, 사용자 지정 지표를 만들 수 있습니다.
- 프로그래밍 방식으로 샘플링 규칙을 만들고 업데이트하세요.

보안 및 ADOT 최적화 계층이 추가된 표준화된 OpenTelemetry SDK를 유연하게 사용하려면 SDK를 사용하세요. AWS AWS Distro for OpenTelemetry (ADOT) SDK는 벤더에 구애받지 않는 패키지로, 코드를 다시 작성할 필요 없이 다른 벤더의 백엔드 및 비 서비스와 통합할 수 있습니다. AWS

이미 X-Ray SDK를 사용하고 있고, AWS 백엔드와만 통합하고, X-Ray나 애플리케이션 코드와의 상호 작용 방식을 바꾸고 싶지 않다면 X-Ray SDK를 사용하십시오.

각 기능에 대한 자세한 내용은 을 참조하십시오. [AWS 배포판용 OpenTelemetry SDK와 X-Ray SDK 중 선택](#)

## ADOTSDK 사용

ADOTSDK는 데이터를 백엔드 서비스로 보내는 오픈소스 API, 라이브러리 및 에이전트 세트입니다. ADOT에서 AWS지원되며 여러 백엔드 및 에이전트와 통합되며 커뮤니티에서 유지 관리하는 수많은 오픈 소스 라이브러리를 제공합니다. OpenTelemetry ADOTSDK를 사용하여 애플리케이션을 계측하고 로그, 메타데이터, 지표 및 추적을 수집할 수 있습니다. 를 ADOT 사용하여 서비스를 모니터링하고의 메트릭을 기반으로 경보를 설정할 수도 있습니다. CloudWatch

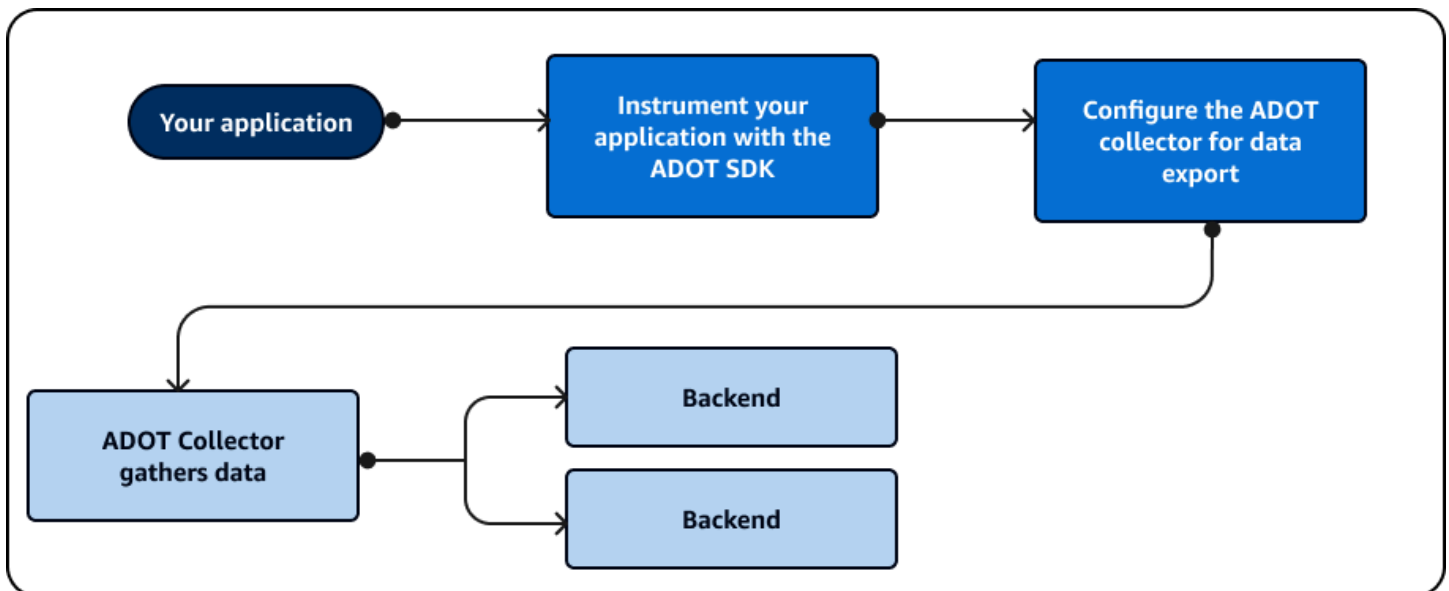
ADOTSDK를 사용하는 경우 에이전트와 함께 사용할 수 있는 옵션은 다음과 같습니다.

- [CloudWatch 에이전트와](#) 함께 ADOT SDK를 사용하는 것이 좋습니다.
- [ADOTCollector와](#) 함께 ADOT SDK를 사용하십시오. 보안 및 최적화 AWS 계층을 갖춘 벤더에 구매 받지 않는 소프트웨어를 사용하려는 경우 권장됩니다.

ADOTSDK를 사용하려면 다음과 같이 하세요.

- ADOTSDK를 사용하여 애플리케이션을 계측하십시오. 자세한 내용은 [ADOT 기술](#) 문서의 프로그래밍 언어 설명서를 참조하십시오.
- ADOT수집기가 수집한 데이터를 어디로 보낼지 알려주도록 수집기를 구성하십시오.

ADOT수집기는 데이터를 수신한 후 ADOT 구성에서 지정한 백엔드로 데이터를 전송합니다. ADOT다음 다이어그램과 같이 외부 공급업체를 비롯한 여러 백엔드에 데이터를 보낼 수 있습니다. AWS



AWS 정기적으로 ADOT 업데이트하여 기능을 추가하고 프레임워크에 [OpenTelemetry](#) 맞게 조정합니다. 업데이트 및 향후 개발 ADOT 계획은 대중이 이용할 수 있는 [로드맵의](#) 일부입니다. ADOT 다음을 포함한 여러 프로그래밍 언어를 지원합니다.

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

Python을 사용하는 경우 응용 프로그램을 자동으로 ADOT 계측할 수 있습니다. 사용을 시작하려면 ADOT [OpenTelemetry Collector용 AWS 배포판 소개 및 사용하기](#)를 참조하십시오.

## SDK 사용 X-Ray

X-Ray SDK는 데이터를 AWS 백엔드 AWS 서비스로 전송하는 API 및 라이브러리 세트입니다. X-Ray SDK를 사용하여 애플리케이션을 계측하고 추적 데이터를 수집하십시오. X-Ray SDK를 사용하여 로그 또는 지표 데이터를 수집할 수 없습니다.

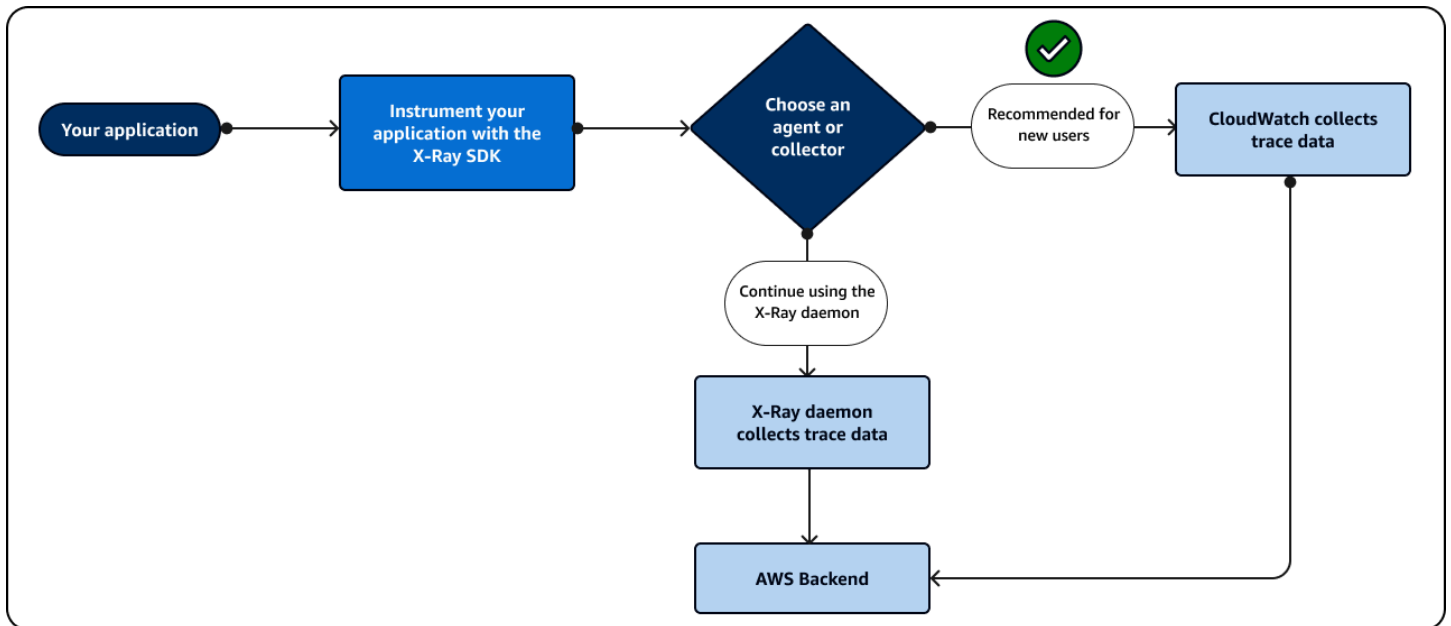
X-Ray SDK를 사용하는 경우 에이전트와 함께 사용할 수 있는 옵션은 다음과 같습니다.

- X-Ray SDK와 함께 사용 [AWS X-Ray 다이몬](#) — 애플리케이션 코드를 업데이트하지 않으려면 이 옵션을 사용하십시오.
- CloudWatch 에이전트와 함께 X-Ray SDK 사용 — (권장) CloudWatch 에이전트는 X-Ray SDK와 호환됩니다.

X-Ray SDK를 사용하려면 다음과 같이 하십시오.

- X-Ray SDK를 사용하여 애플리케이션을 계측하십시오.
- 수집기가 수집한 데이터를 어디로 보낼지 알려주도록 수집기를 구성하십시오. CloudWatch 에이전트 또는 X-Ray 데몬을 사용하여 추적 정보를 수집할 수 있습니다.

수집기 또는 에이전트는 데이터를 수신한 후 에이전트 구성에서 지정한 AWS 백엔드로 데이터를 보냅니다. X-Ray SDK는 다음 다이어그램에 표시된 대로 AWS 백엔드로만 데이터를 전송할 수 있습니다.



를 사용하는 경우 Java, X-Ray SDK를 사용하여 애플리케이션을 자동으로 계측할 수 있습니다. X-Ray SDK 사용을 시작하려면 다음 프로그래밍 언어와 관련된 라이브러리를 참조하십시오.

- [Go](#)
- [Java](#)
- [Node.js](#)
- [Python](#)
- [.NET](#)
- [Ruby](#)

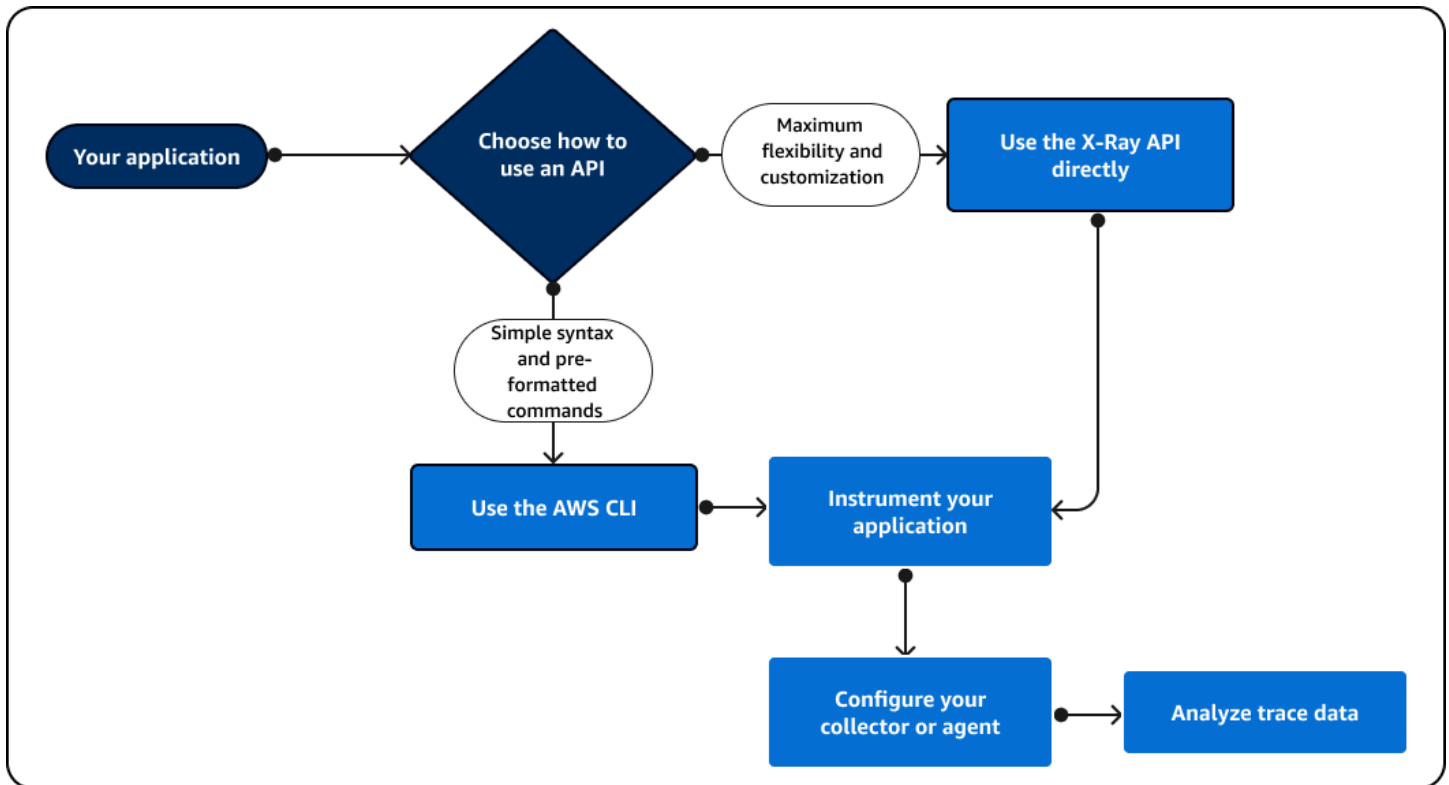
## X-Ray API 사용

X-Ray SDK가 프로그래밍 언어를 지원하지 않는 경우, X-Ray API를 직접 사용하거나 AWS Command Line Interface (AWS CLI) 를 사용하여 X-Ray API 명령을 호출할 수 있습니다. 다음 지침을 사용하여 API와 상호 작용하는 방법을 선택하십시오.

- 미리 형식이 AWS CLI 지정된 명령을 사용하거나 요청 내 옵션과 함께 사용하여 더 간단한 구문을 사용하십시오.
- X-Ray API를 직접 사용하여 X-Ray에 보내는 요청에 대한 유연성을 극대화하고 사용자 지정할 수 있습니다.

대신 [X-Ray API](#)를 직접 사용하는 경우 요청을 올바른 데이터 형식으로 매개변수화해야 하며 인증 및 오류 처리를 구성해야 할 수도 있습니다. AWS CLI

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



X-Ray API를 사용하여 트레이스 데이터를 X-Ray에 직접 전송할 수 있습니다. X-Ray API는 다음과 같은 일반적인 작업을 포함하여 X-Ray SDK에서 사용할 수 있는 모든 기능을 지원합니다.

- [PutTraceSegments](#)— 세그먼트 문서를 X-Ray에 업로드합니다.
- [BatchGetTraces](#)— 추적 ID 목록에서 추적 목록을 검색합니다. 검색된 각 트레이스는 단일 요청으로 이루어진 세그먼트 문서 모음입니다.
- [GetTraceSummaries](#)— 트레이스의 ID 및 주석을 검색합니다. `a`를 `FilterExpression` 지정하여 추적 요약의 하위 집합을 검색할 수 있습니다.
- [GetTraceGraph](#)— 특정 추적 ID에 대한 서비스 그래프를 검색합니다.
- [GetServiceGraph](#)— 들어오는 요청을 처리하고 다운스트림 요청을 호출하는 서비스를 설명하는 JSON 서식이 지정된 문서를 검색합니다.

또한 애플리케이션 코드 내의 AWS Command Line Interface (AWS CLI) 를 사용하여 프로그래밍 방식으로 X-Ray와 상호 작용할 수 있습니다. 는 다른 AWS 서비스기능을 포함하여 X-Ray SDK에서 사용할

수 있는 모든 기능을 AWS CLI 지원합니다. 다음 함수는 앞서 나열한 API 작업의 간단한 형식 버전입니다.

- [put-trace-segments](#)— 세그먼트 문서를 X-Ray에 업로드합니다.
- [batch-get-traces](#)— 추적 ID 목록에서 추적 목록을 검색합니다. 검색된 각 트레이스는 단일 요청으로 이루어진 세그먼트 문서 모음입니다.
- [get-trace-summaries](#)— 트레이스의 ID 및 주석을 검색합니다. a를 FilterExpression 지정하여 추적 요약의 하위 집합을 검색할 수 있습니다.
- [get-trace-graph](#)— 특정 추적 ID에 대한 서비스 그래프를 검색합니다.
- [get-service-graph](#)— 들어오는 요청을 처리하고 다운스트림 요청을 호출하는 서비스를 설명하는 JSON 서식이 지정된 문서를 검색합니다.

시작하려면 운영 체제에 맞게 [AWS CLI](#)를 설치해야 합니다. AWS 지원 Linux macOS 및 Windows 운영 체제. X-Ray 명령 목록에 대한 자세한 내용은 [X-Ray용AWS CLI 명령 참조 안내서](#)를 참조하십시오.

## X-Ray API 살펴보기

X-Ray API는 AWS SDK를 통해 또는 HTTPS를 통해 직접 모든 X-Ray 기능에 액세스할 수 있도록 합니다. AWS Command Line Interface [X-Ray API Reference](#)에는 API 작업의 입력 파라미터, 그리고 해당 입력 파라미터가 반환하는 필드 및 데이터 형식이 설명되어 있습니다.

AWS SDK를 사용하여 X-Ray API를 사용하는 프로그램을 개발할 수 있습니다. X-Ray 콘솔과 X-Ray 데몬은 모두 AWS SDK를 사용하여 X-Ray와 통신합니다. 각 언어의 AWS SDK에는 X-Ray API 작업 및 유형에 매핑되는 클래스 및 메서드에 대한 참조 문서가 있습니다.

### AWS SDK 레퍼런스

- Java – [AWS SDK for Java](#)
- JavaScript – [AWS SDK for JavaScript](#)
- .NET – [AWS SDK for .NET](#)
- Ruby – [AWS SDK for Ruby](#)
- Go – [AWS Go용 SDK](#)
- PHP – [AWS SDK for PHP](#)
- Python – [AWS SDK for Python \(Boto\)](#)

AWS Command Line Interface 는 Python용 SDK를 사용하여 AWS API를 호출하는 명령줄 도구입니다. AWS API를 처음 학습할 때는 사용 가능한 매개변수를 탐색하고 JSON 또는 텍스트 형식으로 서비스 출력을 볼 수 있는 쉬운 방법을 AWS CLI 제공합니다.

`aws xray`하위 명령에 대한 자세한 내용은 [AWS CLI 명령 참조](#)를 참조하십시오.

## AWS CLI와 함께 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 API를 사용할 수 있습니다. 샘플 애플리케이션에는 AWS CLI에서 이러한 API를 사용하는 방법을 보여주는 스크립트가 포함되어 있습니다.

### 필수 조건

이 튜토리얼에서는 Scorekeep 샘플 애플리케이션과 포함된 스크립트를 사용하여 트레이스 데이터와 서비스 맵을 생성합니다. 샘플 애플리케이션 [자습서의](#) 지침에 따라 애플리케이션을 시작하십시오.

이 자습서에서는 AWS CLI 를 사용하여 X-Ray API의 기본 사용법을 보여줍니다. 윈도우, 리눅스, OS-X에서 사용할 수 있는 [AWS CLI](#)는 누구나 퍼블릭 API에 대한 명령줄 액세스를 제공합니다. AWS 서비스

#### Note

샘플 애플리케이션이 생성된 지역과 동일한 지역으로 AWS CLI 구성되었는지 확인해야 합니다.

샘플 애플리케이션 테스트용으로 포함된 스크립트는 cURL을 사용하여 API 및 jq로 트래픽을 보내 출력을 구문 분석합니다. jq 실행 파일은 [stedolan.github.io](https://stedolan.github.io)에서 다운로드할 수 있고, curl 실행 파일은 <https://curl.haxx.se/download.html>에서 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 중단 간 시나리오를 실행하고 보다 다양한 트레이스 데이터를 생성합니다.

### `test-api.sh` 스크립트를 사용하려면

1. [Elastic Beanstalk 콘솔](#)을 엽니다.



2. 사용 중인 환경의 [관리 콘솔](#)로 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":
["JQVLE0M2","D67QLPIC","VF9BM9NC","0EAA6GK9","2A705073","1U2LFTLJ","HUKIDD70","BAN1C8FI","G
["BS8F8LQ","4MTTSPKP","4630ETES","SVEBCL3N","N7CQ1GHP","0840NEPD","EG4BPROQ","V4BLIDJ3","9R
```

## X-Ray API 사용

AWS CLI는 `aws xray`를 포함하여 [GetServiceGraph](#) X-Ray가 제공하는 모든 API 작업에 대한 명령을 제공합니다. [GetTraceSummaries](#) 지원되는 모든 작업과 각 작업에서 사용되는 데이터 형식에 대한 자세한 내용은 [AWS X-Ray API 참조](#)를 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\":\"QAKE0S8DD0LJM245KAOPMA746BVV4KQNS05AEMVJF66Q9ASUAAJG\",
\"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
":"388GER0C4PCA6D59ED3CTI5EEJV4KQNS05AEMVJF66Q9ASUAAJG","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 콘솔](#)을 엽니다.
2. 사용 중인 환경의 [관리 콘솔](#)로 이동합니다.
3. 작업을 선택합니다.
4. [Terminate Environment]를 선택합니다.
5. 종료를 선택합니다.

트레이스 데이터는 30일 후에 X-Ray에서 자동으로 삭제됩니다.

## 트레이스 데이터를 X-Ray로 보내기

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 subsegment의 type과 상위 세그먼트를 식별하는 parent\_id를 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 [X-Ray 세그먼트 문서](#) 단원을 참조하십시오.

## 추적 ID 생성

데이터를 X-Ray로 보내려면 각 요청에 대해 고유한 추적 ID를 생성해야 합니다.

## X-Ray 트레이스 ID 형식

X-Ray trace\_id는 하이픈으로 구분된 3개의 숫자로 구성됩니다. 예를 들어 1-58406520-a006649127e371903a2de979입니다. 여기에는 다음이 포함됩니다.

- 버전 번호, 즉 1.
- 원래 요청 시간을 Unix 시대 시간으로 나타낸 8자리 16진수

예를 들어, 2016년 12월 1일 오전 10시 (PST) 기준 에포크 타임은 초 또는 16진수로 표시됩니다.  
1480615200 58406520

- 24자리 16진수 트레이스를 위한 전 세계적으로 고유한 96비트 식별자입니다.

### Note

이제 X-Ray는 [W3C Trace Context 사양을 준수하는 기타 프레임워크를 사용하여 OpenTelemetry 만든 추적 ID](#)를 지원합니다. W3C 추적 ID는 X-Ray로 전송할 때 X-Ray 추적 ID 형식으로 지정해야 합니다. 예를 들어, W3C 추적 ID는 X-Ray로 전송할 1-4efaaf4d-1e8720b39541901950019ee5 때와 같은 형식으로 4efaaf4d1e8720b39541901950019ee5 지정해야 합니다. X-Ray Trace ID에는 Unix 에포크 타임의 원래 요청 타임스탬프가 포함되지만 W3C 추적 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](#)

사용: PutTraceSegments

[PutTraceSegments](#) API를 사용하여 세그먼트 문서를 업로드할 수 있습니다. 이 API에는 JSON 세그먼트 문서의 목록을 가져오는 단일 파라미터 TraceSegmentDocuments가 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 프로세서와 Windows는 JSON 문자열에서 따옴표를 인용하거나 이스케이프하는 데 필요한 요구 PowerShell 사항이 다릅니다. 자세한 내용은 AWS CLI 사용 설명서의 [문자열 인용](#)을 참조하십시오.

출력에는 처리를 실패한 세그먼트가 모두 나열됩니다. 예를 들어 트레이스 ID의 날씨가 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 AWS로 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}\n가 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 cat을 사용하여 /dev/udp로 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"}
```

[데몬\(daemon\) 로그](#)를 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)
```

X-Ray에서 데이터 가져오기

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

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 +%s는 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 `groupARN`를 포함합니다. 다음의 예는 `Example1`이라는 그룹에 서비스 그래프를 호출하는 방법을 보여줍니다.

Example 그룹 `Example1`의 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` 검색에서는 `TraceID` 시간을 사용하고 계산된 범위 내에서 시작된 트레이스를 반환합니다. `[start_time, end_time)` 이 타임스탬프 범위는 내의 타임스탬프 인코딩을 기반으로 계산되거나 수동으로 정의할 수 있습니다. `TraceId`
- **이벤트 시간** — 시간 경과에 따라 발생하는 이벤트를 검색하기 위해 AWS X-Ray에서는 이벤트 타임스탬프를 사용하여 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 시간을 사용하여 과거 1~2분 사이의 추적 요약 목록을 가져옵니다. `Traceld`

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": "01a"
        }
      }
    ]
  },
  "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
\": \"MFQ8CGJ3JTDDVVVASUAAJGQ6NJ82F738B0B4KQNS05AEMVJF66Q9\", \"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\": \"0la\" } }, { \"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-6B829NT9V509\\"",\\"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\":"
\\"MFQ8CGJ3JTDDVVVASUAAJGQ6NJ82F738B0B4KQNS05AEMVJF66Q9\","\\"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-6B829NT9V509\\"",
\\"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. 자세한 정보는 [X-Ray 세그먼트 문서](#)를 참조하세요.

또한 X-Ray는 세그먼트를 직접 전송하지 않는 서비스에 대한 다운로드 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 추적 맵의 DynamoDB 리소스를 나타내는 추론된 세그먼트를 만들고 추적 문서에 추가합니다.

API에서 여러 트레이스를 가져오려면 [AWS CLI 쿼리](#)를 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 API](#)로 추적 요약을 생성하면 부분 추적 요약을 JSON 형식으로 재사용하여 근본 원인을 기반으로 정제된 필터 표현식을 생성할 수 있습니다. 구체화 단계에 대한 연습은 아래 예제를 참조하십시오.

Example 예제 GetTraceSummaries 출력 - 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 식을 사용한 쿼리에 대한 자세한 내용은 [Explore the X-Ray 콘솔의 필터 표현식 사용 섹션](#)을 참조하십시오.

### Example JSON 필터 예제

```

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

```

### X-Ray API를 사용하여 샘플링, 그룹 및 암호화 설정 구성

X-Ray는 샘플링 규칙, 그룹 규칙 및 암호화 설정을 구성하기 위한 API를 제공합니다.

#### 암호화 설정

암호화에 사용할 AWS Key Management Service (AWS KMS) 키를 지정하는 [PutEncryptionConfig](#)에 사용합니다.

#### 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.

[GetEncryptionConfig](#)를 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. 샘플링에 대한 자세한 내용은 [을 참조하십시오 오샘플링 규칙을 구성합니다.](#) 태그 추가 및 관리에 대한 자세한 내용은 [X-Ray 샘플링 규칙 및 그룹 태그 지정하기](#) 섹션을 참조하세요.

[GetSamplingRules](#)로 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 [UpdateSamplingRule](#)로 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 [UntagResource](#)를 사용하십시오.

```

$ 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
},

```

[CreateSamplingRule](#)로 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 [DeleteSamplingRule](#)을 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
}
}

```

## 그룹

X-Ray API를 이용해 계정의 그룹을 관리할 수 있습니다. 그룹은 필터 표현식으로 정의한 추적 모음입니다. 그룹을 사용하여 추가 서비스 그래프를 생성하고 Amazon CloudWatch 지표를 제공할 수 있습니다. X-Ray API를 통한 서비스 그래프 및 지표 작업에 대한 자세한 내용은 [X-Ray에서 데이터 가져오기](#) 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.

CreateGroup이(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}"
}

```

GetGroups로 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 [UntagResource](#)를 사용하십시오.

```

$ 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

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 [GetSamplingRules](#)로 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 [X-Ray API를 사용하여 샘플링, 그룹 및 암호화 설정 구성](#)을 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 [GetSamplingTargets](#)에 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. UnprocessedStatistics에는 규칙이 삭제되었거나, 입력의 통계 문서가 너무 오래되었음을 나타내는 오류 또는 권한 오류가 포함될 수 있습니다.

## X-Ray 세그먼트 문서

트레이스 세그먼트는 애플리케이션이 처리하는 요청의 JSON 표현입니다. 트레이스 세그먼트는 원 요청에 관한 정보, 애플리케이션이 로컬로 처리하는 작업 정보를 기록하며, 애플리케이션이 AWS 리소스, HTTP API와 SQL 데이터베이스에 대해 적용하는 다운스트림 호출에 대한 정보가 있는 하위 세그먼트 정보도 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 추적 맵의 다운스트림 서비스와 리소스를 나타냅니다.

X-Ray는 세그먼트 문서를 위한 JSON 스키마를 제공합니다. [스키마는 1.0.0에서 다운로드할 수 있습니다.](#) [xray-segmentdocument-schema-v](#) 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 정보는 [필터 표현식 사용](#)을 참조하세요.

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 정보는 [세그먼트 문서를 X 대몬\(daemon\)으로 전송](#)을 참조하세요.

## 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. 예를 들어 `1-58406520-a006649127e371903a2de979`입니다. 여기에는 다음이 포함됩니다.

- 버전 번호, 즉 1.
- 원래 요청 시간을 Unix 시대 시간으로 나타낸 8자리 16진수

예를 들어, 2016년 12월 1일 오전 10시 (PST) 기준 에포크 타임은 초 또는 16진수로 표시됩니다.  
`1480615200 58406520`

- 24자리 16진수 트레이스를 위한 전 세계적으로 고유한 96비트 식별자입니다.

**Note**

이제 X-Ray는 [W3C Trace Context 사양을 준수하는 기타 프레임워크를 사용하여 OpenTelemetry 만든 추적 ID](#)를 지원합니다. W3C 추적 ID는 X-Ray로 전송할 때 X-Ray 추적 ID 형식으로 지정해야 합니다. 예를 들어, W3C 추적 ID는 X-Ray로 전송할 1-4efaaf4d-1e8720b39541901950019ee5 때와 같은 형식으로 4efaaf4d1e8720b39541901950019ee5 지정해야 합니다. X-Ray Trace ID에는 Unix 에 포크 타임의 원래 요청 타임스탬프가 포함되지만 W3C 추적 ID를 X-Ray 형식으로 전송할 때는 필요하지 않습니다.

**트레이스 ID 보안**

트레이스 ID는 [응답 헤더](#)에 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: 1480615200.010 또는 1.480615200010E9. 소수점 자리는 필요한 만큼 사용하십시오. 가능하다면 마이크로초 해상도를 권장합니다.
- `end_time` – 세그먼트를 닫은 시간을 표시한 숫자입니다. 예: 1480615200.090 또는 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 `name`은 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 유형.

### 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::Environment`로 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 헤더](#)에 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 만들어 AWS SDK로 AWS 서비스 호출한 리소스, 내부 또는 외부 HTTP 웹 API에 대한 호출 또는 SQL 데이터베이스 쿼리를 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 type의 subsegment, trace\_id와 상위 세그먼트를 식별하는 parent\_id를 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: 1480615200.010 또는 1.480615200010E9.
- `end_time` – 하위 세그먼트를 닫은 시간을 표시한 숫자입니다. 예: 1480615200.090 또는 1.480615200090E9. `end_time` 또는 `in_progress`를 지정합니다.



- `in_progress` – 하위 세그먼트가 시작되었지만 완료되지 않았음을 기록하고자 `end_time`을 지정하는 대신 `true`로 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_id`는 하이픈으로 구분된 3개의 숫자로 구성됩니다. 예를 들어 `1-58406520-a006649127e371903a2de979`입니다. 여기에는 다음이 포함됩니다.

- 버전 번호, 즉 1.
- 원래 요청 시간을 Unix 시대 시간으로 나타낸 8자리 16진수

예를 들어, 2016년 12월 1일 오전 10시 (PST) 기준 에포크 타임은 초 또는 16진수로 표시됩니다.  
`1480615200 58406520`

- 24자리 16진수 트레이스를 위한 전 세계적으로 고유한 96비트 식별자입니다.

#### Note

이제 X-Ray는 [W3C Trace Context 사양을 준수하는 기타 프레임워크를 사용하여 OpenTelemetry 만든 추적 ID](#)를 지원합니다. W3C 추적 ID는 X-Ray로 전송할 때 X-Ray 추적 ID 형식으로 지정해야 합니다. 예를 들어, W3C 추적 ID는 X-Ray로 전송할 `1-4efaaf4d-1e8720b39541901950019ee5` 때와 같은 형식으로 `4efaaf4d1e8720b39541901950019ee5` 지정해야 합니다. X-Ray Trace ID에는 Unix 에포크 타임의 원래 요청 타임스탬프가 포함되지만 W3C 추적 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` – 애플리케이션에서 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. 예를 들어 GET입니다.
  - `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 Address에서, 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 트race된 서비스를 대상으로 함을 표시하는 부울입니다. 이 필드가 true로 설정되면, X-Ray는 다운스트림 서비스가 이 블록을 포함한 하위 세그먼트의 `id`와 일치하는 `parent_id`가 있는 세그먼트를 업로드할 때까지는 트race가 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
}

```

### 주석

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 (다음 예시) 는 AWS제공된 SDK 및 debug 클라이언트에서 사용하도록 AWS. 예약되어 있습니다.

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": "3AIENM5J4ELQ3SP0DHKBIRVIC3VV4KQNS05AEMVJF66Q9ASUAAJG",
        "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 AWS 계정에 전송한다면, 애플리케이션을 실행하는 계정 ID를 기록합니다.
- `cloudwatch_logs`— 단일 CloudWatch 로그 그룹을 설명하는 개체 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 플러그인으로 차단

```
"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"
  }
}
```

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



## aws (하위 세그먼트)

모든 필드는 선택 사항입니다.

- `operation` – AWS 서비스 나 리소스를 상대로 호출한 API 작업의 이름입니다.
- `account_id`— 애플리케이션이 다른 계정의 리소스에 액세스하거나 세그먼트를 다른 계정으로 전송하는 경우 애플리케이션이 액세스한 AWS 리소스를 소유한 계정의 ID를 기록하십시오.
- `region` – 리소스가 애플리케이션과 다른 리전에 있다면, 리전을 기록합니다. 예를 들어 `us-west-2`입니다.
- `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": "UBQNS05AEM8T4FDA4RQDEB940VTDRVV4K4HIRGVJF66Q9ASUAAJG",
  }
}
```

## 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 true로 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 Error를 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** – stack에서 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 다이몬

## Note

이제 CloudWatch 에이전트를 사용하여 Amazon EC2 인스턴스 및 온프레미스 서버에서 지표, 로그 및 추적을 수집할 수 있습니다. CloudWatch 에이전트 버전 1.300025.0 이상에서는 또는 [OpenTelemetry](#)X-Ray 클라이언트 SDK에서 트레이스를 수집하여 [X-Ray](#)로 보낼 수 있습니다. AWS Distro for OpenTelemetry (ADOT) Collector 또는 X-Ray 데몬 대신 CloudWatch 에이전트를 사용하여 트레이스를 수집하면 관리하는 에이전트 수를 줄이는 데 도움이 될 수 있습니다. 자세한 내용은 CloudWatch 사용 설명서의 [CloudWatch 에이전트](#) 항목을 참조하십시오.

AWS X-Ray 데몬은 UDP 포트 2000에서 트래픽을 수신하고 원시 세그먼트 데이터를 수집하여 API로 전달하는 소프트웨어 애플리케이션입니다. AWS X-Ray 데몬은 SDK와 함께 작동하며 AWS X-Ray SDK에서 보낸 데이터가 X-Ray 서비스에 도달할 수 있도록 실행 중이어야 합니다. X-Ray 데몬(daemon)은 오픈 소스 프로젝트입니다. [github.com/aws/](https://github.com/aws/)에서 프로젝트를 팔로우하고 이슈와 풀 리퀘스트를 제출할 수 있습니다. [GitHub aws-xray-daemon](#)

AWS Lambda 그리고 AWS Elastic Beanstalk 해당 서비스와 X-Ray의 통합을 사용하여 데몬을 실행하십시오. Lambda는 샘플링된 요청에 대해 함수가 호출될 때마다 자동으로 데몬(daemon)을 실행합니다. Elastic Beanstalk에서는 [XRayEnabled 구성 옵션을 사용](#)하여 환경 내 인스턴스에서 데몬(daemon)을 실행합니다. 자세한 내용을 알아보려면 다음 섹션을 참조하세요.

X-Ray 데몬을 로컬, 온-프레미스 또는 다른 AWS 서비스곳에서 실행하려면 다운로드하여 [실행한](#) 다음 세그먼트 문서를 X-Ray에 업로드할 수 있는 [권한을 부여하십시오](#).

## 데몬 다운로드

Amazon S3, Amazon ECR 또는 Docker Hub에서 데몬(daemon)을 다운로드한 다음 로컬에서 실행하거나 시작 시 Amazon EC2 인스턴스에 설치할 수 있습니다.

### Amazon S3

X-Ray 데몬(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 데몬(daemon)의 최신 3.x 릴리스를 가리킵니다. 특정 릴리스를 다운로드하려면 다음과 같이 하십시오.

- 버전 이전 릴리스를 3.3.0 다운로드하려면 버전 3.x 번호로 바꾸십시오. 예를 들어 2.1.0입니다. 버전 3.3.0 이전에는 사용할 수 있는 아키텍처만 arm64 있습니다.
- 버전별로 릴리스를 3.3.0 다운로드하려면 버전 번호와 arch 아키텍처 유형으로 3.x 바꾸십시오. 예: 2.1.0 및 arm64.

X-Ray 자산은 지원되는 모든 리전에서 버킷에 복제됩니다. 사용자 또는 AWS 리소스에서 가장 가까운 버킷을 사용하려면 위 링크의 지역을 해당 지역으로 바꾸세요.

```
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 ECR](#)에서 데몬(daemon)을 찾을 수 있습니다. 이미지를 가져오기 전에 Amazon ECR 퍼블릭 레지스트리에 [docker 클라이언트를 인증](#)해야 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.x을 alpha 또는 특정 버전 번호로 바꾸어 다운로드할 수 있습니다.

프로덕션 환경에서는 알파 태그가 있는 데몬 이미지를 사용하지 않는 것이 좋습니다.

## Docker Hub

대몬(daemon)은 [Docker Hub](#)에서 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.x을 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 [GnuPG](#)를 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 `-n`으로 리전을 설정합니다.

```
~/Downloads$ ./xray -o -n us-east-2
```

플랫폼별 세부 지침은 다음 항목을 참조하십시오.

- Linux (로컬) – [Linux에서 X-Ray 데몬\(daemon\) 실행](#)
- Windows (로컬) – [Windows에서 X-Ray 데몬\(daemon\) 실행](#)
- Elastic Beanstalk – [AWS Elastic Beanstalk에서 X-Ray 데몬\(daemon\) 실행](#)
- Amazon EC2 – [Amazon EC2에서 X-Ray 데몬 실행하기](#)
- Amazon ECS – [Amazon ECS에서 X-Ray 데몬\(daemon\) 실행하기](#)

명령줄 옵션 또는 구성 파일을 사용하여 데몬의 동작을 추가로 사용자 지정할 수 있습니다. 세부 정보는 [AWS X-Ray 데몬 설정하기](#)를 참조하세요.

## 데몬(daemon)에 X-Ray로 데이터를 전송할 권한 부여

X-Ray 데몬은 AWS SDK를 사용하여 추적 데이터를 X-Ray에 업로드하며, 이를 위해서는 권한이 있는 AWS 자격 증명이 필요합니다.



Amazon EC2에서는 대몬(daemon)이 자동으로 인스턴스의 인스턴스 프로파일 역할을 사용합니다. [데몬을 로컬에서 실행하는 데 필요한 자격 증명에 대한 자세한 내용은 로컬 애플리케이션 실행을 참조하십시오.](#)

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 실행](#)을 참조하십시오.

자세한 정보는 [ID 및 액세스 관리 대상 AWS X-Ray](#)을 참조하세요.

## 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 설정하기](#) 섹션을 참조하세요.

Elastic Beanstalk에서는 플랫폼에 따라 대몬(daemon) 로그의 위치가 설정됩니다. 세부 정보는 [AWS Elastic Beanstalk에서 X-Ray 대몬\(daemon\) 실행](#)를 참조하세요.

## AWS X-Ray 데몬 설정하기

명령줄 옵션 또는 구성 파일을 사용하여 X-Ray 데몬(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.

데몬(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
```

데몬(daemon)을 Amazon EC2가 아니라 로컬로 실행 중인 경우, `-o` 옵션을 추가해 인스턴스 프로파일 자격 증명 확인을 건너뛰도록 하여 데몬(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.

데몬(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](#)
- [명령줄 옵션 사용](#)
- [구성 파일 사용](#)

### 지원되는 환경 변수

X-Ray 데몬(daemon)은 다음과 같은 환경 변수를 지원합니다.

- `AWS_REGION` – X-Ray 서비스 엔드포인트의 [AWS 리전](#)을 지정합니다.
- `HTTPS_PROXY` – 세그먼트 업로드에 이용할 데몬(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).
- `-p, --proxy-address` — 프록시를 AWS X-Ray 통해 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, --version` — AWS X-Ray 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 문서 업로드를 위한 AWS X-Ray 최대 동시 호출 수입니다.
- `Region` — 특정 지역의 AWS X-Ray 서비스에 세그먼트를 전송합니다.
- `Socket` — 데몬(daemon)의 바인딩을 구성합니다.
  - `UDPAddress` — 데몬(daemon)이 수신 대기하는 포트를 변경합니다.
  - `TCPAddress` — 다른 TCP 포트에서 [X-Ray 서비스에 대한 호출](#)을 수신 대기합니다.
- `Logging` — 로깅 동작을 구성합니다.
  - `LogRotation` — 로그 순환을 비활성화하려면 `false`로 설정합니다.
  - `LogLevel` — 로그 수준을 가장 자세한 수준에서 가장 낮은 수준으로 변경: `dev`, `info` 또는 `debug`, `prodwarn`, `error`. `prod` 기본값은 `prod`이며, `info`와 동일합니다.
  - `LogPath` — 지정된 파일 경로로 로그를 출력합니다.

- LocalMode – EC2 인스턴스 메타데이터 확인을 건너뛰려면 true로 설정합니다.
- ResourceARN— 데몬을 실행하는 리소스의 Amazon AWS 리소스 이름 (ARN).
- RoleARN – 지정된 IAM 역할을 맡아 세그먼트를 다른 계정으로 업로드합니다.
- ProxyAddress— 프록시를 AWS X-Ray 통해 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

AWS X-Ray 데몬은 리눅스, macOS, Windows 또는 Docker 컨테이너에서 로컬로 실행할 수 있습니다. 구성된 애플리케이션을 개발 및 테스트할 때 대몬(daemon)을 실행해 X-Ray로 추적 데이터를 전달합니다. [여기](#)의 지침에 따라 데몬을 다운로드한 다음 추출합니다.

로컬에서 실행하는 경우 데몬은 사용자 디렉터리의 AWS SDK 자격 증명 파일 또는 환경 변수에서 자격 증명을 읽을 수 있습니다. `.aws/credentials` 자세한 정보는 [대몬\(daemon\)에 X-Ray로 데이터를 전송할 권한 부여](#)를 참조하세요.

이 데몬은 포트 2000에서 UDP 데이터를 수신 대기합니다. 구성 파일 및 명령줄 옵션을 사용하여 포트 및 기타 옵션을 변경할 수 있습니다. 자세한 정보는 [AWS X-Ray 데몬 설정하기](#)를 참조하세요.

## Linux에서 X-Ray 데몬(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 &
```

`pkill`은 배경에서 실행되는 데몬 프로세스를 종료합니다.

```
~$ pkill xray
```

## (Docker 컨테이너에서 X-Ray 데몬(daemon) 실행

도커 컨테이너에서 데몬을 로컬로 실행하기 위해 다음 텍스트를 `Dockerfile`이라는 파일로 저장합니다. Amazon ECR에서 전체 [이미지 예제](#)를 다운로드하십시오. 자세한 내용은 [데몬\(daemon\) 다운로드하기](#)를 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 build`를 사용하여 컨테이너 이미지를 빌드합니다.

```
~/xray-daemon$ docker build -t xray-daemon .
```

`docker run`을 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`— 컨테이너에 `.aws` 디렉터리에 대한 읽기 전용 액세스 권한을 부여하여 SDK 자격 증명을 읽을 수 있도록 합니다. AWS
- `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 [명령줄](#)에서 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 만들고 실행합니다.

Example PowerShell 스크립트 - 윈도우

```
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 데몬(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 &
```

## AWS Elastic Beanstalk에서 X-Ray 대몬(daemon) 실행

트레이스 데이터를 애플리케이션에서 AWS X-Ray로 중계하기 위해 Elastic Beanstalk 환경의 Amazon EC2 인스턴스에서 X-Ray 대몬(daemon)을 실행할 수 있습니다. 지원되는 플랫폼 목록은 AWS Elastic Beanstalk 개발자 안내서의 [AWS X-Ray 디버깅 구성](#)을 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 사용할 가능한 데몬의 버전을 확인하려면 [지원되는 플랫폼 항목](#)을 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 콘솔](#)을 엽니다.
2. 사용 중인 환경의 [관리 콘솔](#)로 이동합니다.
3. Configuration(구성)을 선택합니다.

4. [Software Settings]를 선택합니다.
5. [X-Ray daemon]에서 [Enabled]를 선택합니다.
6. Apply(적용)를 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 Beanstalk은 AWS Management Console 또는 명령줄에서 인스턴스 로그를 가져오기 위한 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:\Program Files\Amazon\XRay\logs\xray-service.log
```

자세한 내용은 AWS Elastic Beanstalk 개발자 안내서의 [Elastic Beanstalk 환경에서 Amazon EC2 인스턴스의 로그 보기](#)를 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](#) 섹션을 참조하세요.

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 Amazon 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 이미지](#)를 제공합니다. 자세한 내용은 [대몬\(daemon\) 다운로드하기](#)를 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](#) 섹션을 참조하세요.

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 모드](#)를 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 CMD가 업데이트됩니다. 자세한 내용은 [Docker 실행 참조](#)를 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 정의(Task Definitions)를 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 쉽표로 구분된 명령줄 옵션 목록을 명령 필드에 추가합니다.
7. 업데이트를 선택합니다.
8. 정보를 확인한 후 생성(Create)을 선택합니다.

다음 예제에서는 RoleARN 옵션에 대해 쉽표로 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 사용할 수 있는 명령줄 옵션에 대한 자세한 내용은 [AWS X-Ray 대몬\(daemon\) 구성](#)을 참조하십시오.

## 애플리케이션 측정 대상 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 빌드된 계측을 추가할 수 있습니다.
- 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

- [배포판을 사용하여 애플리케이션을 계측하기 AWS OpenTelemetry](#)
- [AWS X-Ray SDK로 애플리케이션 계측하기](#)
- [AWS 배포판용 OpenTelemetry SDK와 X-Ray SDK 중 선택](#)
- [다음과 같이 애플리케이션을 계측하십시오 Go](#)
- [다음과 같이 애플리케이션을 계측하십시오 Java](#)
- [다음과 같이 애플리케이션을 계측하십시오 Node.js](#)
- [다음과 같이 애플리케이션을 계측하십시오 Python](#)
- [다음과 같이 애플리케이션을 계측하십시오 .NET](#)
- [Ruby로 애플리케이션을 계측하세요](#)

## 배포판을 사용하여 애플리케이션을 계측하기 AWS OpenTelemetry

AWS 배포판 OpenTelemetry (ADOT) 은 클라우드 네이티브 컴퓨팅 재단 (CNCF) 프로젝트를 기반으로 하는 AWS 배포판입니다. OpenTelemetry OpenTelemetry 분산된 트레이스 및 메트릭을 수집할 수 있는 단일 오픈 소스 API, 라이브러리 및 에이전트 세트를 제공합니다. 이 툴킷은 에서 테스트, 최적화,

보안 및 지원하는 SDK, 자동 계측 에이전트 및 수집기를 포함한 업스트림 OpenTelemetry 구성 요소를 배포한 것입니다. AWS

ADOT를 사용하면 엔지니어가 애플리케이션을 한 번 계측하여 상관 관계가 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. 하나는 X-Ray와 함께 사용할 수 있는 OpenTelemetry SDK이고 다른 하나는 X-Ray와 함께 사용할 수 있는 OpenTelemetry Collector용 AWS 배포판입니다. OpenTelemetry [with AWS X-Ray](#) 및 [AWS 서비스기타용 AWS 배포판 사용에 대한 자세한 내용은 설명서용 배포판을 참조하십시오.](#) [AWS OpenTelemetry](#)

언어 지원 및 사용에 대한 자세한 내용은 [AWS Observability](#) on을 참조하십시오. GitHub

#### Note

이제 CloudWatch 에이전트를 사용하여 Amazon EC2 인스턴스 및 온프레미스 서버에서 지표, 로그 및 추적을 수집할 수 있습니다. CloudWatch 에이전트 버전 1.300025.0 이상에서는 또는 [OpenTelemetryX-Ray](#) 클라이언트 SDK에서 트레이스를 수집하여 [X-Ray로](#) 보낼 수 있습니다. AWS Distro for OpenTelemetry (ADOT) Collector 또는 X-Ray 데몬 대신 CloudWatch 에이전트를 사용하여 트레이스를 수집하면 관리하는 에이전트 수를 줄이는 데 도움이 될 수 있습니다. 자세한 내용은 CloudWatch 사용 설명서의 [CloudWatch 에이전트](#) 항목을 참조하십시오.

ADOT에는 다음이 포함됩니다:

- [AWS Go용 OpenTelemetry 배포판](#)
- [AWS 자바용 배포판 OpenTelemetry](#)
- [AWS 배포판 대상 OpenTelemetry JavaScript](#)
- [AWS Python용 OpenTelemetry 배포판](#)
- [AWS .NET용 배포판 OpenTelemetry](#)

현재 ADOT에는 [Java](#) 및 [Python](#)에 대한 자동 계측 지원이 포함되어 있습니다. 또한 [ADOT를 사용하면 ADOT 관리형 AWS Lambda 레이어를 통해 Java, Node.js 및 Python 런타임을 사용하여 Lambda 함수 및 해당 다운스트림 요청을 자동으로 계측할 수 있습니다.](#)

Java 및 Go용 ADOT SDK는 X-Ray 통합 샘플링 규칙을 지원합니다. 다른 언어의 X-Ray 샘플링 규칙에 대한 지원이 필요한 경우 AWS X-Ray SDK 사용을 고려해 보세요.

**Note**

이제 W3C 추적 ID를 X-Ray로 보낼 수 있습니다. 기본적으로 로 만든 트레이스는 [W3C 트레이스 컨텍스트 사양을 기반으로 하는 트레이스 ID](#) 형식을 OpenTelemetry 갖습니다. 이는 X-Ray SDK를 사용하거나 X-Ray와 통합된 AWS 서비스에 의해 생성되는 추적 ID의 형식과는 다릅니다. W3C 형식의 트레이스 ID가 X-Ray에서 허용되도록 하려면 [ADOT AWS X-Ray 컬렉터 버전 0.34.0 이상에 포함된 익스포터](#) 버전 0.86.0 이상을 사용해야 합니다. 이전 버전의 익스포터는 트레이스 ID 타임스탬프를 검증하므로 W3C 트레이스 ID가 거부될 수 있습니다.

## AWS X-Ray SDK로 애플리케이션 계측하기

AWS X-Ray 애플리케이션이 트레이스를 X-Ray로 전송하도록 계측하기 위한 언어별 SDK 세트가 포함되어 있습니다. 각 X-Ray SDK는 다음을 제공합니다:

- 코드에 인터셉터를 추가하여 수신 HTTP 요청을 추적할 수 있습니다.
- 애플리케이션이 다른 클라이언트를 호출하는 데 사용하는 AWS SDK 클라이언트를 계측하기 위한 클라이언트 핸들러 AWS 서비스
- 다른 내부 및 외부 HTTP 웹 서비스에 대한 호출을 계측하기 위한 HTTP 클라이언트

또한 X-Ray SDK는 SQL 데이터베이스에 대한 계측 호출, 자동 AWS SDK 클라이언트 계측 및 기타 기능을 지원합니다. 트레이스 데이터를 직접 X-Ray로 전송하는 대신, SDK는 UDP 트래픽을 수신 대기하는 데몬(daemon) 프로세스로 JSON 세그먼트 문서를 전송합니다. [X-Ray 데몬\(daemon\)](#)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.js SDK용 SDK](#)
- [AWS X-Ray Python용 SDK](#)
- [AWS X-Ray SDK for .NET](#)
- [AWS X-Ray SDK for Ruby](#)

현재 X-Ray에는 [Java](#)에 대한 자동 계측 지원이 포함되어 있습니다.

## AWS 배포판용 OpenTelemetry SDK와 X-Ray SDK 중 선택

X-Ray에 포함된 SDK는 AWS에서 제공하는 긴밀하게 통합된 계측 솔루션의 일부입니다. AWS Distro for OpenTelemetry 는 X-Ray가 수많은 추적 솔루션 중 하나일 뿐인 광범위한 산업 솔루션의 일부입니다. 두 방법 중 하나를 사용하여 X-Ray에서 end-to-end 추적을 구현할 수 있지만 가장 유용한 접근 방식을 결정하려면 차이점을 이해하는 것이 중요합니다.

다음에 필요한 OpenTelemetry 경우 AWS 배포판을 사용하여 애플리케이션을 계측하는 것이 좋습니다.

- 코드를 다시 작성할 필요 없이 여러 추적 백엔드로 트레이스를 전송할 수 있는 기능
- 커뮤니티에서 유지 관리하는 각 언어에 대한 수많은 라이브러리 도구 지원 OpenTelemetry
- 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 배포판은 Lambda 함수 계측을 위한 더 간단한 시작 환경을 OpenTelemetry 제공합니다. 그러나 유연성 OpenTelemetry 제공으로 인해 Lambda 함수에 추가 메모리가 필요하고 호출 시 콜드 스타트 지연 시간이 증가하여 추가 요금이 발생할 수 있습니다. 지연 시간이 짧고 동적으로 구성 가능한 백엔드 대상과 같은 고급 기능이 필요하지 OpenTelemetry 않은 경우 AWS X-Ray SDK를 사용하여 애플리케이션을 계측하는 것이 좋습니다.

다음에 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)

## 다음과 같이 애플리케이션을 계측하십시오 Go

트레이스를 X-Ray로 전송하도록 Go 애플리케이션을 계측하는 방법에는 두 가지가 있습니다.

- [AWS 배포판 대상 OpenTelemetry Go — Collector](#)용 AWS 배포판을 통해 Amazon CloudWatch AWS X-Ray, Amazon OpenSearch Service를 비롯한 여러 AWS 모니터링 솔루션으로 상호 관

련된 지표와 추적을 전송할 수 있는 오픈 소스 라이브러리 세트를 제공하는 [AWS 배포판입니다](#).  
OpenTelemetry

- [AWS X-Ray SDK for Go](#) — X-Ray [데몬을](#) 통해 트레이스를 생성하고 X-Ray로 전송하기 위한 라이브러리 세트입니다.

자세한 내용은 [AWS 배포판용 OpenTelemetry SDK와 X-Ray SDK 중 선택을\(를\)](#) 참조하세요.

## AWS Distro for OpenTelemetry Go

OpenTelemetry Go용 AWS 배포판을 사용하면 애플리케이션을 한 번 계측하여 상호 관련된 지표와 추적을 Amazon CloudWatch, AWS X-Ray, Amazon OpenSearch Service를 포함한 여러 AWS 모니터링 솔루션으로 전송할 수 있습니다. OpenTelemetry용 AWS 배포판과 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 위한 AWS 배포판입니다.

시작하려면 [OpenTelemetry Go용 AWS 배포판 설명서](#)를 참조하십시오.

AWS X-Ray 및 다른 AWS 서비스 버전과 함께 OpenTelemetry용 AWS 배포판을 사용하는 방법에 대한 자세한 내용은 [OpenTelemetry용 AWS 배포판](#) 또는 [OpenTelemetry용 AWS 배포판 설명서](#)를 참조하십시오.

언어 지원 및 사용법에 대한 자세한 내용은 [Github의 AWS 관찰성](#)을 참조하세요.

## Go용 AWS X-Ray SDK

Go용 X-Ray SDK는 트레이스 데이터를 생성하고 X-Ray 데몬(daemon)에 보내기 위한 클래스 및 메서드를 제공하는 Go 애플리케이션용 라이브러리 집합입니다. 트레이스 데이터는 애플리케이션이 처리하는 수신 HTTP 요청, 그리고 애플리케이션이 AWS SDK, HTTP 클라이언트 또는 SQL 데이터베이스 커넥터를 사용하여 다운스트림 서비스에 하는 호출에 대한 정보를 포함합니다. 또한 수동으로 세그먼트를 생성하고 디버그 정보를 주석 및 메타데이터에 추가할 수도 있습니다.

go get을 사용하여 해당 [GitHub 리포지토리](#)에서 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 [세그먼트](#)를 생성하고, 응답이 전송되면 세그먼트를 완료합니다. 세그먼트가 열려 있는 동안에는 SDK 클라이언트의 메서드를 이용해 정보를 세그먼트에 추가하고 하위



세그먼트를 만들어 다운스트림 호출을 트race할 수 있습니다. 또한 SDK는 세그먼트가 열려 있는 동안 애플리케이션에서 발생하는 예외를 자동으로 기록합니다.

계측되는 애플리케이션 또는 서비스에 의해 호출되는 Lambda 함수의 경우, Lambda는 [추적 헤더](#)를 읽고 샘플링된 요청을 자동으로 추적합니다. 그 밖의 함수의 경우 수신 요청을 샘플링 및 추적하도록 [Lambda를 구성](#)합니다. 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 정보는 [AWS Lambda](#) 그리고 [AWS X-Ray](#) 섹션을 참조하세요.

다음으로 [클라이언트를 AWS 함수에 대한 호출로 래핑](#)합니다. 이 단계를 통해 X-Ray가 모든 클라이언트 메서드에 대한 호출을 구성할 수 있습니다. 또한 [SQL 데이터베이스에 대한 호출을 구성](#)할 수 있습니다.

SDK를 사용하기 시작한 후에, [레코더와 미들웨어를 구성](#)하여 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 메타데이터](#)에서 하는 작업을 기록합니다. 주석은 [필터 표현식](#)과 함께 사용할 수 있도록 인덱싱된 단순한 키 값 쌍이기 때문에, 특정 데이터를 포함한 트race를 검색할 수 있습니다. 메타데이터 항목은 제한이 적으며 JSON으로 직렬화할 수 있는 모든 객체와 어레이를 기록할 수 있습니다.

### 주석 및 메타데이터

주석 및 메타데이터는 X-Ray 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 세그먼트](#)의 클라이언트 호출

을 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.

- Go용 AWS SDK 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

Configure을 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](#)
- [샘플링 규칙](#)
- [로깅](#)
- [환경 변수](#)

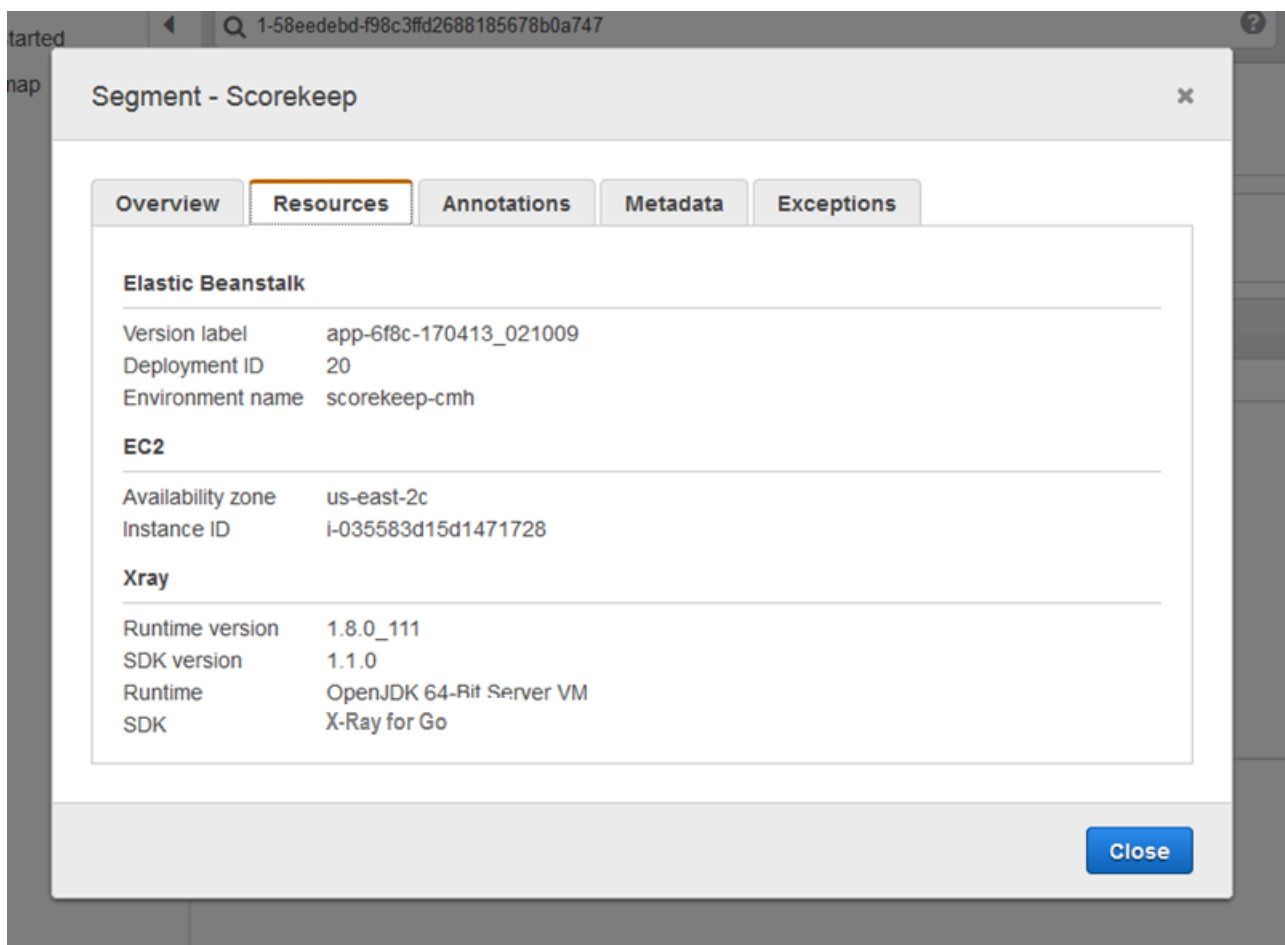
- [configure 사용](#)

## 서비스 플러그인

plugins을 사용하여 애플리케이션을 호스팅하는 서비스에 대한 정보를 기록할 수 있습니다.

### 플러그인

- Amazon EC2 — 인스턴스 ID, 가용 영역 및 CloudWatch 로그 그룹을 EC2Plugin 추가합니다.
- Elastic Beanstalk – ElasticBeanstalkPlugin이 환경 이름, 버전 레이블 및 배포 ID를 추가합니다.
- Amazon ECS — ECSPlugin이 컨테이너 ID를 추가합니다.



플러그인을 사용하려면 다음 패키지 중 하나를 가져옵니다.

```
"github.com/aws/aws-xray-sdk-go/awspplugins/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`와 같은 해결 순서를 사용하여 출처를 결정합니다.

### 샘플링 규칙

SDK는 X-Ray 콘솔에서 정의하는 샘플링 규칙을 사용하여 기록할 요청을 결정합니다. 기본 규칙은 매 초 첫 번째 요청을 추적하고, 모든 서비스에서 추가 요청의 5%를 X-Ray로 추적 전송합니다. [X-Ray 콘솔에서 추가 규칙을 생성](#)하여 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 X-Ray에 도달하여 샘플링 규칙을 가져올 수 없는 경우 1초가 시작될 때 수신되는 첫 번째 요청의 기본 로컬 규칙과 호스트당 추가 요청의 5%로 되돌아갑니다. 호스트가 샘플링 API

를 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.

아니요 AWS Lambda, 샘플링 속도는 수정할 수 없습니다. 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 대기](#)하도록 대몬(daemon)을 구성한 경우 또는 다른 호스트에서 실행 중인 경우 이 변수를 사용합니다.

환경 변수는 코드에 설정된 동등한 값을 재정의합니다.

### configure 사용

Configure 메서드를 사용하여 Go용 X-Ray SDK를 구성할 수도 있습니다. Configure는 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.json에 정의된 전략을 가져오는 LocalizedSamplingStrategy를 사용합니다.

## StreamingStrategy

이 StreamingStrategy 객체는 true가 RequiresStreaming 반환될 때 세그먼트를 스트리밍할지 여부를 지정합니다. 지정하지 않으면 X-Ray는 하위 세그먼트 수가 20보다 크면 샘플링된 세그먼트를 스트림하는 DefaultStreamingStrategy를 사용합니다.

## ExceptionFormattingStrategy

이 ExceptionFormattingStrategy 객체는 여러 예외를 처리하는 방법을 지정합니다. 지정하지 않으면 X-Ray는 DefaultExceptionFormattingStrategy를 error 유형의 XrayError, 오류 메시지 및 스택 추적과 함께 사용합니다.

## Go용 X-Ray SDK로 수신 HTTP 요청 계측하기

Go용 X-Ray SDK를 사용하여 애플리케이션이 Amazon EC2, AWS Elastic Beanstalk 또는 Amazon ECS의 EC2 인스턴스에서 처리하는 수신 HTTP 요청을 트레이스할 수 있습니다.

xray.Handler를 사용하여 수신 HTTP 요청을 구성합니다. Go용 X-Ray SDK는 xray.Handler 클래스에서 표준 Go 라이브러리 http.Handler 인터페이스를 구현하여 웹 요청을 가로칩니다. xray.Handler 클래스는 요청의 컨텍스트를 사용하고, 수신 헤더를 구문 분석하고, 필요한 경우 응답 헤더를 추가하여 xray.Capture로 제공된 http.Handler를 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 샘플링되는 각 요청마다 세그먼트 하나를 생성합니다. 자세한 정보는 [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 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.

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-Ray는 서비스 이름을 사용하여 사용자의 애플리케이션을 식별하고, 이를 다른 애플리케이션, 데이터베이스, 외부 API, 사용자의 애플리케이션에서 사용하는 AWS 리소스와 구별합니다. X-Ray SDK는 수신 요청에 대한 세그먼트를 생성할 때 해당 세그먼트의 [이름 필드](#)에 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 변수](#)를 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 MyApp을 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 추적하기

[애플리케이션이 데이터를 저장하거나, 대기열에 기록하거나, 알림을 보내기 위해 를 호출하는 경우, X-Ray SDK for Go는 통화를 하위 세그먼트로 다운스트림으로 추적합니다. AWS 서비스](#) 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 [핸들러](#)에 전달된 `http.Request` 객체에서 `context`를 전달합니다.

### Example main.go — SDK 호출 AWS

```

func listTablesWithContext(ctx context.Context) {
    output := dynamo.ListTablesWithContext(ctx, &dynamodb.ListTablesInput{})
    doSomething(output)
}

```

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 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": "UBQNS05AEM8T4FDA4RQDEB940VTDRVV4K4HIRGVJF66Q9ASUAAJG",
  }
}
```

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-client`는 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 [세그먼트](#)를 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 {
```

```

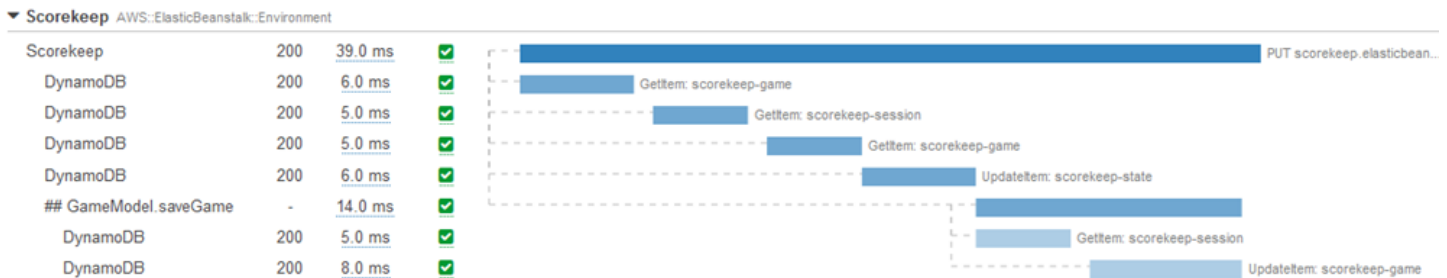
var err error

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

주석과 메타데이터를 사용하여 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 표현식](#)에서 사용하기 위해 인덱싱됩니다. 주석은 콘솔의 트레이스를 그룹화할 때 사용할 데이터를 기록하거나 [GetTraceSummaries](#) API를 호출할 때 사용하십시오.

메타데이터는 객체 및 목록을 포함한 모든 유형의 값을 가질 수 있는 키-값 페어지만, 필터 표현식에 사용할 수 있도록 인덱싱되지 않습니다. 트레이스에 저장하고 싶지만 검색에는 사용하지 않을 추가 데이터는 메타데이터를 사용하여 기록하십시오.

세그먼트에는 주석과 메타데이터 외에 [사용자 ID 문자열](#)도 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. 밑줄 기호 (\_) 이외의 공백이나 기호는 사용할 수 없습니다.
- 값 — X-Ray 주석의 값은 최대 1,000개의 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 `annotations.key` 키워드를 [필터 표현식](#)에 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 setUser를 호출합니다.

```
mySegment.User = "U12345"
```

사용자 ID의 트레이스를 찾으려면, user 키워드를 [필터 표현식](#)에 적용하십시오.

## 다음과 같이 애플리케이션을 계측하십시오 Java

트레이스를 X-Ray로 전송하도록 Java 애플리케이션을 계측하는 방법에는 두 가지가 있습니다.

- [AWS 배포판 대상 OpenTelemetry Java — Collector용](#) AWS 배포판을 통해 Amazon CloudWatch AWS X-Ray, Amazon OpenSearch Service를 비롯한 여러 AWS 모니터링 솔루션으로 상호 관련된 지표와 추적을 전송할 수 있는 오픈 소스 라이브러리 세트를 제공하는 [AWS 배포판입니다](#). OpenTelemetry
- [AWS X-Ray SDK for Java](#) — X-Ray [데몬](#)을 통해 트레이스를 생성하고 X-Ray로 전송하기 위한 라이브러리 세트입니다.

자세한 내용은 [AWS 배포판용 OpenTelemetry SDK와 X-Ray SDK 중 선택](#)을(를) 참조하세요.

## AWS Distro for OpenTelemetry Java

Java용 OpenTelemetry AWS 배포판 (ADOT)을 사용하면 애플리케이션을 한 번 계측하여 상호 관련된 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 위한 AWS 배포판입니다. ADOT Java에는 자동 계측 지원이 포함되어 애플리케이션에서 코드 변경 없이 트레이스를 전송할 수 있습니다.

시작하려면 [Java용 OpenTelemetry AWS 배포판 설명서](#)를 참조하십시오.

AWS X-Ray 및 다른 AWS 서비스 버전과 함께 OpenTelemetry AWS 배포판을 사용하는 방법에 대한 자세한 내용은 [OpenTelemetry용 AWS 배포판](#) 또는 [OpenTelemetry용 AWS 배포판 설명서](#)를 참조하십시오.



언어 지원 및 사용법에 대한 자세한 내용은 [Github의 AWS 관찰성](#)을 참조하세요.

## AWS X-Ray SDK를 위한 Java

Java용 X-Ray SDK는 추적 데이터를 생성하고 X-Ray 데몬으로 전송하기 위한 클래스와 메서드를 제공하는 웹 애플리케이션용 라이브러리 Java 세트입니다. 추적 데이터에는 응용 프로그램에서 제공하는 수신 HTTP 요청 및 응용 프로그램이 AWS SDK, HTTP 클라이언트 또는 SQL 데이터베이스 커넥터를 사용하여 다운스트림 서비스에 보내는 호출에 대한 정보가 포함됩니다. 또한 수동으로 세그먼트를 생성하고 디버그 정보를 주석 및 메타데이터에 추가할 수도 있습니다.

Java용 X-Ray SDK는 오픈 소스 프로젝트입니다. [github.com/aws/에서 프로젝트를 팔로우하고 이슈와 풀 리퀘스트를 제출할 수 있습니다](https://github.com/aws/aws-xray-sdk-java) [GitHub. aws-xray-sdk-java](https://github.com/aws-xray-sdk-java)

[AWSXRayServletFilter](#)를 서블릿 필터로 추가하여 수신 요청 트레이스를 시작합니다. 서블릿 필터가 [세그먼트](#)를 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, 애플리케이션 런타임에 코드를 추가하지 않고도 애플리케이션이 실행되는 AWS동안 애플리케이션을 계측할 수 있습니다.

다음으로, Java용 X-Ray SDK를 사용하여 빌드 구성에 SDK AWS SDK for Java [인스트루먼트 하위 모듈을 포함하여 클라이언트를 계측합니다](#). 인스트루먼트된 클라이언트를 사용하여 다운스트림 AWS 서비스 또는 리소스를 호출할 때마다 SDK는 해당 호출에 대한 정보를 하위 세그먼트에 기록합니다. AWS 서비스 또한 서비스 내에서 액세스하는 리소스는 추적 맵에 다운스트림 노드로 표시되어 개별 연결의 오류 및 제한 문제를 식별하는 데 도움이 됩니다.

모든 다운스트림 통화를 계측하지 않으려면 Instrumentor 하위 모듈을 생략하고 AWS 서비스계측할 클라이언트를 선택하면 됩니다. SDK 서비스 클라이언트에 [a를 추가하여](#) 개별 클라이언트를 TracingHandler 계측하십시오. AWS

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 HTTPClientBuilder](#)를 사용하여 Apache HTTP 클라이언트를 계측할 수 있습니다. SQL 쿼리를 구성하려면 [SDK의 인터셉터를 데이터 원본에 추가](#)합니다.

SDK를 사용하기 시작한 후에, [레코더와 servlet 필터를 구성](#)하여 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 메타데이터](#)에서 하는 작업을 기록합니다. 주석은 [필터 표현식](#)과 함께 사용할 수 있도록 인덱싱된 단순한 키 값 쌍이기 때문에, 특정 데이터를 포함한 트레 이스를 검색할 수 있습니다. 메타데이터 항목은 제한이 적으며 JSON으로 직렬화할 수 있는 모든 객체 와 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 세그먼트](#)의 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 클라이언트를 요청 AWS 서비스 핸들러로 추가하여 AWS SDK for Java 클라이언트와의 통화를 계속합니다.
- [aws-xray-recorder-sdk-aws-sdk-v2](#)— 추적 클라이언트를 요청 AWS 서비스 인터셉터로 추가하여 AWS SDK for Java 2.2 이상 클라이언트와의 전화 통화를 계속합니다.
- [aws-xray-recorder-sdk-aws-sdk-instrumentor](#)— 를 사용하면 모든 `aws-xray-recorder-sdk-aws-sdk` AWS SDK for Java 클라이언트를 자동으로 계속합니다.
- [aws-xray-recorder-sdk-aws-sdk-v2-instrumentor](#)— 모든 AWS SDK for Java 2.2 이상 클라이언트를 자동으로 계속합니다. `aws-xray-recorder-sdk-aws-sdk-v2`

- [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 CloudWatch Amazon 지표를 게시할 수 있습니다.

Maven 또는 Gradle을 사용하여 애플리케이션을 빌드하는 경우, [빌드 구성에 Java용 X-Ray SDK를 추가합니다.](#)

SDK의 클래스 및 메서드에 대한 참조 문서는 API 참조용 [AWS X-Ray SDK](#)를 참조하십시오. Java

## 요구 사항

Java용 X-Ray Java SDK에는 8개 이상의 서블릿 API 3, SDK 및 AWS 잭슨이 필요합니다.

SDK는 컴파일 및 실행 시간 시 다음 라이브러리에 의존합니다.

- AWS 버전 1.11.398 이상용 Java SDK
- 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 애플리케이션](#)을 참조하십시오.

## Java용 AWS X-Ray 자동 계측 에이전트

Java용 AWS X-Ray 자동 계측 에이전트는 최소한의 개발 노력으로 Java 웹 애플리케이션을 계측하는 추적 솔루션입니다. 이 에이전트는 서버릿 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 서버리스 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 Agent를 사용하기 위해 DiSCo를 이해할 필요는 없지만 [홈페이지를 GitHub](#) 방문하여 프로젝트에 대해 자세히 알아볼 수 있습니다. 또한 X-Ray 에이전트는 완전히 오픈 소스입니다. 소스 코드를 보거나, 기여를 하거나, 에이전트에 대한 문제를 제기하려면 해당 [저장소를](#) 방문하십시오. GitHub

## 샘플 애플리케이션

[eb-java-scorekeep](#) 샘플 애플리케이션은 X-Ray Agent와 함께 측정하도록 조정되었습니다. 이러한 기능은 에이전트가 수행하므로 이 브랜치에는 서버리스 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 튜토리얼](#)에 나와 있습니다.

## 시작하기

자체 애플리케이션에서 X-Ray 자동 계측 Java 에이전트를 시작하려면 다음 절차를 따르십시오.

1. 사용 중인 환경에서 X-Ray 대몬(daemon)을 실행합니다. 자세한 내용은 [X-Ray 대몬\(daemon\)](#)을 참조하십시오.
2. [에이전트의 최신 배포판](#)을 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
```

- 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.
- 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](#)를 참조하십시오. 모든 필드는 선택 사항입니다.

속성 이름	유형	유효값	설명	환경 변수	시스템 속성	기본값
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



속성 이름	유형	유효값	설명	환경 변수	시스템 속성	기본값
samplingStrategy	String	CENTRAL, LOCAL, NONE, ALL	에이전트가 사용하는 샘플링 전략. ALL은 모든 요청을 캡처하고, NONE은 요청을 캡처하지 않습니다. <a href="#">샘플링 규칙</a> 을 참조하세요.	N/A	N/A	CENTRAL
traceInjectionFix	String	모든 문자열	로그에 트race ID를 삽입하기 전에 제공된 접두사를 포함합니다.	N/A	N/A	없음 (빈 문자열)

속성 이름	유형	유효값	설명	환경 변수	시스템 속성	기본값
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	<a href="#">DefaultSamplingRules.json</a>
awsServiceHandlerManifest	String	절대 파일 경로	AWS SDK 클라이언트로부터 추가 정보를 캡처하는 사용자 지정 매개변수 허용 목록의 경로입니다.	N/A	N/A	<a href="#">DefaultOperationParameterWhitelist.json</a>

속성 이름	유형	유효값	설명	환경 변수	시스템 속성	기본값
awsSdkVersion	Integer	1, 2	사용 중인 <a href="#">Java용AWS SDK</a> 버전입니다. awsServiceHandlerManifest 이 설정되지 않은 경우 무시됩니다.	N/A	N/A	2
maxStackTrace길이	정수	음수가 아닌 정수	트레이스에 기록할 스택 트레이스의 최대 줄 수입니다.	N/A	N/A	50
streamingThreshold	정수	음수가 아닌 정수	최소한 이 만큼의 서버 세그먼트가 닫힌 후에는 청크가 너무 커지지 않도록 해당 서버 세그먼트가 out-of-band 데몬으로 스트리밍됩니다.	N/A	N/A	100

속성 이름	유형	유효값	설명	환경 변수	시스템 속성	기본값
traceIdInjection	불	True, False	<a href="#">로깅 구성</a> 에 설명된 종속성 및 구성도 추가된 경우 로그에 X-Ray trace ID 삽입을 활성화합니다. 그렇지 않으면 아무 작업도 수행하지 않습니다.	N/A	N/A	TRUE
pluginsEnabled	불	True, False	운영 중인 AWS 환경에 대한 메타데이터를 기록하는 플러그인을 활성화합니다. <a href="#">플러그인</a> 을 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	기본값
contextPropagation	불	True, False	true인 경우 스레드 간에 X-Ray 컨텍스트를 자동으로 전파합니다. 그렇지 않으면 Thread Local을 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 [로깅](#)을 참조하십시오.

## 수동 구성

에이전트의 자동 계측 외에도 수동 계측을 수행하려는 경우 프로젝트에 종속 항목으로 X-Ray SDK를 추가하세요. [수신 요청 추적](#)에서 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 세그먼트](#)를 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\(daemon\) 설명서](#)를 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 올바르게 따랐는지 확인하십시오.

애플리케이션 로그에 “AWS X-Ray 컨텍스트 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 만들지 못했기 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 로그](#)를 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 계측](#)의 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 SDK로 AWS Lambda 함수를 한 번 호출하거나 일부 HTTP 엔드포인트를 주기적으로 폴링하는 등 “실행만 하면 잊어버리는” 백그라운드 스레드를 사용하여 작업을 수행하는 경우, 스레드 간에 컨텍스트를 전파하는 동안 에이전트가 혼란을 야기할 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 false로 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 [에서 샘플링 규칙 구성을 참조하십시오. X-Ray 콘솔 살펴보기](#)

## Java용 X-Ray SDK 구성

Java용 X-Ray SDK에는 전역 레코더를 제공하는 AWSXRay라는 클래스가 있습니다. 이는 코드를 구성하는 데 사용할 수 있는 TracingHandler입니다. 전역 레코더를 구성하여 수신 HTTP 호출에 대해 세그먼트를 생성하는 AWSXRayServletFilter를 사용자 지정할 수 있습니다.

## Sections

- [서비스 플러그인](#)
- [샘플링 규칙](#)
- [로깅](#)
- [세그먼트 리스너](#)
- [환경 변수](#)
- [시스템 속성](#)

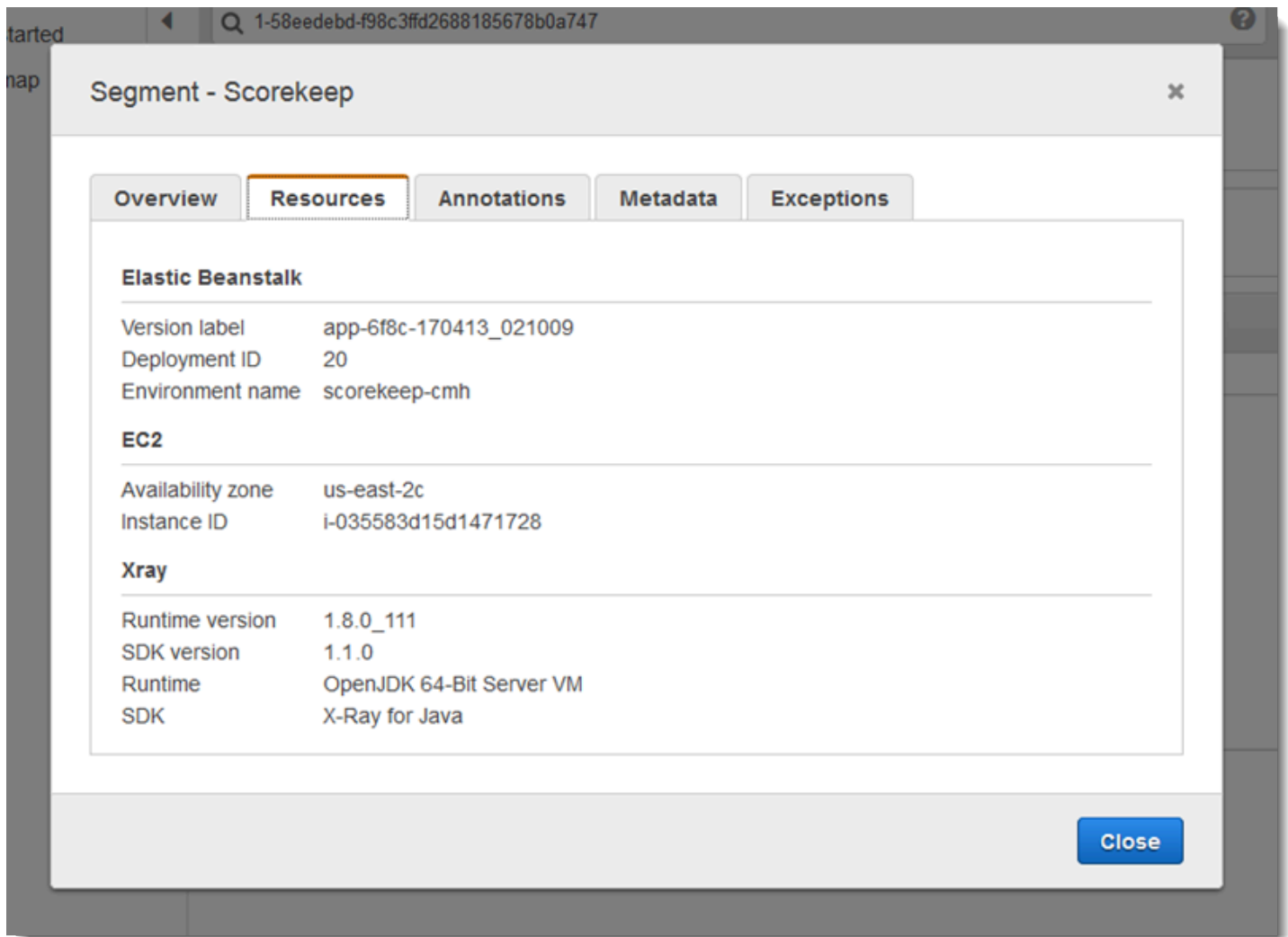
### 서비스 플러그인

plugins을 사용하여 애플리케이션을 호스팅하는 서비스에 대한 정보를 기록할 수 있습니다.

### 플러그인

- Amazon EC2 — 인스턴스 ID, 가용 영역 및 CloudWatch 로그 그룹을 EC2Plugin 추가합니다.
- Elastic Beanstalk – ElasticBeanstalkPlugin이 환경 이름, 버전 레이블 및 배포 ID를 추가합니다.
- Amazon ECS — ECSPlugin이 컨테이너 ID를 추가합니다.
- Amazon EKS — 컨테이너 ID, 클러스터 이름, 포드 ID 및 CloudWatch 로그 그룹을 EKSPPlugin 추가합니다.





플러그인을 사용하려면 `AWSXRayRecorderBuilder`에서 `withPlugin`을 호출하십시오.

Example `src/main/java/scorekeep/ .java WebConfig` - 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 리소스 유형을 나타냅니다. AWS 여러 플러그인을 사용하는 경우 SDK는 ElasticBeanstalk > EKS > ECS > EC2와 같은 해결 순서를 사용하여 출처를 결정합니다.

### 샘플링 규칙

SDK는 X-Ray 콘솔에서 정의하는 샘플링 규칙을 사용하여 기록할 요청을 결정합니다. 기본 규칙은 매 초 첫 번째 요청을 추적하고, 모든 서비스에서 추가 요청의 5%를 X-Ray로 추적 전송합니다. [X-Ray 콘솔에서 추가 규칙을 생성](#)하여 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 X-Ray에 도달하여 샘플링 규칙을 가져올 수 없는 경우 1초가 시작될 때 수신되는 첫 번째 요청의 기본 로컬 규칙과 호스트당 추가 요청의 5%로 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.

아니오 AWS Lambda, 샘플링 속도는 수정할 수 없습니다. 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 `WebConfigsrc/main/java/myapp/.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 ServletContextListener를 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 FATAL입니다. 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 생성](#)할 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 계측된 Java 애플리케이션에서 작동하며 다음과 같은 로깅 구성을 지원합니다.

- Logback 백엔드가 있는 SLF4J 프런트 엔드 API
- Log4J2 백엔드가 있는 SLF4J 프런트 엔드 API

- Log4J2 백엔드가 있는 Log4J2 프론트 엔드 API

각 프론트 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. AWSXRayRecorder를 구축할 때 withSegmentListener 메서드를 포함시킵니다. 이렇게 하면 새 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 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 구성](#)을 참조하십시오.
2. 이후 로깅 문에 트레이스 ID를 삽입하려면 patternLayout의 아무 곳이나 %X{AWS-XRAY-TRACE-ID}를 삽입합니다. %X{}는 MDC에서 제공된 키를 사용해 값을 검색하고 있음을 나타냅니다. PatternLayouts Logback에 대한 자세한 내용은 [PatternLayout](#)을 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 [구성](#)을 참조하십시오.

프로그래밍 방식으로 Log4J2를 구성하는 방법에 대한 자세한 내용은 [프로그래밍 방식으로 구성](#)을 참조하십시오.

- 이후 로깅 문에 트레이스 ID를 삽입하려면 `PatternLayout`의 아무 곳이나 `%X{AWS-XRAY-TRACE-ID}`를 삽입합니다. `%X{}`는 MDC에서 제공된 키를 사용해 값을 검색하고 있음을 나타냅니다. `PatternLayouts` [Log4J2에 대한 자세한 내용은 패턴 레이아웃을 참조하십시오](#).

## 트레이스 ID 주입 예제

다음은 트레이스 ID를 포함하도록 수정된 `PatternLayout` 문자열을 보여줍니다. 트레이스 ID는 스택드 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 쉽게 파싱할 수 있도록 로그문에 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 나타내는 지 확인하고, 그렇다면 추가적인 메타데이터를 추가합니다.

전체 `SegmentListener` 함수 목록을 보려면 [AWS X-Ray X-Ray Recorder SDK for Java API](#)에 대한 설명서를 참조하십시오.



다음 예제에서는 [onBeginSubsegment](#)를 사용하여 생성할 때 모든 하위 세그먼트에 일관된 주석을 추가하고 [afterEndSegment](#)를 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 `AWSXRayRecorder`를 빌드할 때 참조됩니다.

#### Example AWSXRayRecorderBuilder 성명서

```
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_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\_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 대기](#)하도록 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 그룹 이름을 애플리케이션과 관련된 로그 그룹으로 설정합니다. 로그 그룹이 애플리케이션과 동일한 AWS 계정 및 지역을 사용하는 경우, X-Ray는 이 지정된 로그 그룹을 사용하여 애플리케이션의 세그먼트 데이터를 자동으로 검색합니다. 로그 그룹에 대한 자세한 내용은 [로그 그룹 및 스트림 작업을 참조하십시오](#).
- AWS\_XRAY\_TRACING\_NAME – SDK가 세그먼트에 사용할 서비스 이름을 설정합니다. 서블릿 필터의 [세그먼트 이름 지정 전략](#)에 설정한 서비스 이름을 재정의합니다.

환경 변수는 코드에 설정된 동등한 [시스템 속성](#) 및 값을 재정의합니다.

### 시스템 속성

시스템 속성을 [환경 변수](#)에 대한 JVM 지정 대체로 사용할 수 있습니다. SDK는 다음 속성을 지원합니다.

- com.amazonaws.xray.strategy.tracingName – AWS\_XRAY\_TRACING\_NAME와 동등합니다.
- com.amazonaws.xray.emitters.daemonAddress – AWS\_XRAY\_DAEMON\_ADDRESS와 동등합니다.
- com.amazonaws.xray.strategy.contextMissingStrategy – AWS\_XRAY\_CONTEXT\_MISSING와 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 또는 Amazon ECS의 EC2 인스턴스에서 처리하는 수신 HTTP 요청을 추적할 수 있습니다. AWS Elastic Beanstalk

Filter를 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 정보는 [AWS Lambda](#) 그리고 [AWS X-Ray](#)을 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-agent입니다.
- 콘텐츠 길이 — 응답의 content-length입니다.

## Sections

- [애플리케이션에 추적 필터 추가\(Tomcat\)](#)
- [애플리케이션에 추적 필터 추가\(Spring\)](#)
- [세그먼트 이름 지정 전략 구성](#)

### 애플리케이션에 추적 필터 추가(Tomcat)

Tomcat의 경우, <filter>를 프로젝트의 web.xml 파일에 추가합니다. fixedName 파라미터로 수신 요청을 위해 생성된 세그먼트에 적용할 [서비스 이름](#)을 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, Filter를 WebConfig 클래스에 추가합니다. 세그먼트 이름을 하나의 문자열로 [AWSXRayServletFilter](#) 생성자에 보냅니다.

#### Example src/main/java/myapp/.java - 스프링 WebConfig

```
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 서비스 이름을 사용하여 애플리케이션을 식별하고 애플리케이션에서 사용하는 다른 애플리케이션, 데이터베이스, 외부 API 및 리소스와 구별합니다. AWS X-Ray SDK는 수신 요청에 대한 세그먼트를 생성할 때 해당 세그먼트의 [이름 필드](#)에 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 단원](#)에 나온 것처럼 서블릿 필터를 초기화할 때 애플리케이션 이름을 지정하십시오. 이는 수정자를 `SegmentNamingStrategy.fixed()` 호출하고 [AWSXRayServletFilter](#) 생성자에 [SegmentNamingStrategy](#) 전달하여 수정 버전을 만드는 것과 같은 효과가 있습니다.

### Note

코드에 정의한 기본 서비스 이름을 `AWS_XRAY_TRACING_NAME` [환경 변수](#)를 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, 를 [SegmentNamingStrategy](#)호출하여 `SegmentNamingStrategy.dynamic()` 동적을 만든 다음 `AWSXRayServletFilter` 생성자에 전달하십시오.

Example WebConfigsrc/main/java/myapp/ .java - 동적 네이밍을 사용하는 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 추적하기

[애플리케이션이 AWS 서비스 호출하여 데이터를 저장하거나, 대기열에 기록하거나, 알림을 보내는 경우, X-Ray SDK for Java는 호출을 하위 세그먼트로 다운스트림으로 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.

빌드에서 `aws-sdk` 및 `aws-sdk-instrumentor` [하위 모듈](#)을 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 클라이언트에 `XRayClient as`를 추가하십시오. `TracingHandler`

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": "UBQNS05AEM8T4FDA4RQDEB940VTDRVV4K4HIRGVJF66Q9ASUAAJG",
  }
}
```

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 이상에서 다운스트림 호출을 계속하려면 AWS 서비스 빌드 구성에서 `aws-xray-recorder-sdk-aws-sdk-v2-instrumentor` 모듈을 생략할 수 있습니다. 대신 `aws-xray-recorder-sdk-aws-sdk-v2` module을 포함한 다음, `TracingInterceptor`로 구성하여 개별 클라이언트를 계속합니다.

### Example AWS SDK for Java 2.2 이상 - 트레이싱 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 SDK에는 발신 `HttpClientBuilder` HTTP 호출을 계측하는 데 사용할 수 있는 `HttpComponents Apache SDK` 대신 사용할 수 있는 클래스가 `DefaultHttpClient` 포함되어 있습니다.

- `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 계속할 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-sql](#)을 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

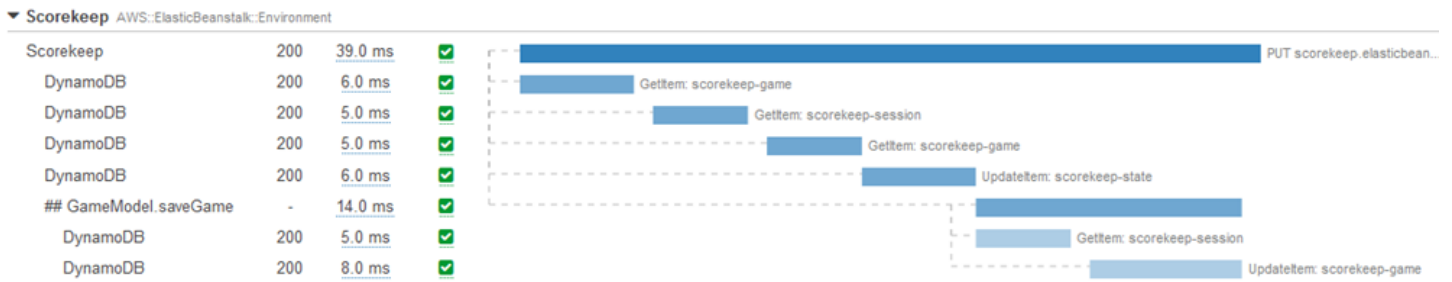
하위 세그먼트는 추적의 [세그먼트](#)를 확장하여 요청을 처리하기 위해 완료된 작업에 대한 세부 정보를 표시합니다. 계속되는 클라이언트에서 직접 호출할 때마다, X-Ray SDK는 하위 세그먼트 안에 생성된 정보를 기록합니다. 추가 하위 세그먼트를 생성하여 다른 하위 세그먼트를 그룹화하거나, 코드 섹션의 성능을 평가하거나, 주석 및 메타데이터를 기록할 수 있습니다.

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 예제에서 하위 세그먼트 내의 코드는 세션 모델의 메서드를 사용하여 DynamoDB에서 게임 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

주석과 메타데이터를 사용하여 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 표현식](#)에서 사용하기 위해 인덱싱됩니다. 주석은 콘솔의 트레이스를 그룹화할 때 사용할 데이터를 기록하거나 [GetTraceSummaries](#) API를 호출할 때 사용하십시오.

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 문자열](#)도 기록할 수 있습니다. 사용자 ID는 세그먼트의 별도 필드에 기록되면 검색용으로 인덱스되지 않습니다.

### Sections

- [Java용 X-Ray SDK로 주석 기록하기](#)



- [Java용 X-Ray SDK로 메타데이터 기록하기](#)
- [Java용 X-Ray SDK로 사용자 ID 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. 밑줄 기호 (\_) 이외의 공백이나 기호는 사용할 수 없습니다.
- 값 — X-Ray 주석의 값은 최대 1,000개의 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();
```

또는

```
import com.amazonaws.xray.AWSXRay;
import com.amazonaws.xray.entities.Subsegment;
...
Subsegment document = AWSXRay.getCurrentSubsegment();
```

2. 문자열 키, 부울, 숫자 또는 문자열 값으로 putAnnotation을 호출합니다.

```
document.putAnnotation("mykey", "my value");
```

SDK는 세그먼트 문서의 annotations 객체에 주석을 키-값 페어로 기록합니다. 같은 키로 putAnnotation을 두 번 호출하면 같은 세그먼트나 하위 세그먼트에 기록했던 값을 덮어씁니다.

특정 값을 포함한 주석이 있는 트레이스를 찾으려면 `annotations.key` 키워드를 [필터 표현식](#)에 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();
```

또는

```
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");
```

또는

키와 값만 이용해 putMetadata를 호출합니다.

```
document.putMetadata("my key", "my value");
```

네임스페이스를 지정하지 않으면, SDK는 default를 사용합니다. 같은 키로 putMetadata을 두 번 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 호출합니다.

```
document.setUser("U12345");
```

컨트롤러에서 setUser를 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/.java — 사용자 ID MoveController](#)

```

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 표현식](#)에 적용하십시오.

## AWS X-Ray Java용 X-Ray SDK의 메트릭스

이 항목에서는 AWS X-Ray 네임스페이스, 지표 및 차원에 대해 설명합니다. Java용 X-Ray SDK를 사용하여 수집된 X-Ray 세그먼트에서 샘플링되지 않은 CloudWatch Amazon 메트릭을 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.

CloudWatch 지표 리포지토리입니다. 지표는 데이터 요소의 기본 CloudWatch 개념이며 시간에 따라 정렬된 데이터 요소 집합을 나타냅니다. 지표 데이터 포인트를 CloudWatch 게시하거나 AWS 서비스해당 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.

에 대한 CloudWatch 자세한 내용은 [Amazon CloudWatch 사용 설명서](#)를 참조하십시오.

### X-Ray CloudWatch 메트릭스

ServiceMetrics/SDK 네임스페이스에 포함된 지표는 다음과 같습니다.

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 시간을 나타냅니다. 개수는 호출 수를 나타냅니다.	밀리초
ErrorRate	평균, 합계	4xx Client Error 상태로 실패하여 오류가 발생한 요청 비율입니다.	%

지표	사용 가능 통계	설명	단위
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.	%

## X선 CloudWatch 치수

다음 표의 측정기준을 사용하여 X-Ray 계측 Java 애플리케이션에 대해 반환되는 메트릭을 구체화하십시오.

측정기준	설명
ServiceType	서비스 유형 (예: AWS::EC2::Instance 또는 알 수 없는 경우 NONE)입니다.
ServiceName	서비스의 정식 이름입니다.

## X-Ray CloudWatch 지표 활성화

다음 절차를 사용하여 계측 Java 애플리케이션에서 트레이스 메트릭을 활성화하십시오.

트레이스 지표를 구성하려면

1. `aws-xray-recorder-sdk-metrics` 패키지를 Apache Maven 종속 항목으로 추가합니다. 자세한 내용은 [Java 서브모듈용 X-Ray SDK](#)를 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());
    }
}
```

- Amazon Elastic Compute Cloud (Amazon EC2), 아마존 Elastic Container Service (Amazon ECS) 또는 아마존 Elastic Kubernetes 서비스 (Amazon EKS) 를 사용하여 지표를 수집하도록 CloudWatch 에이전트를 배포합니다.
  - Amazon EC2를 구성하려면 에이전트 [설치를 참조하십시오. CloudWatch](#)
  - Amazon ECS를 구성하려면 컨테이너 인사이트를 [사용하여 Amazon ECS 컨테이너 모니터링을](#) 참조하십시오.
  - Amazon EKS를 구성하려면 [Amazon CloudWatch Observability EKS 추가 기능을 사용하여 CloudWatch 에이전트 설치를](#) 참조하십시오.
- 에이전트와 통신하도록 SDK를 구성합니다. CloudWatch 기본적으로 SDK는 주소의 CloudWatch 에이전트와 통신합니다. 127.0.0.1 환경 변수 또는 Java 속성을 address:port로 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. <https://console.aws.amazon.com/cloudwatch/> 에서 AWS Management Console 로그인하고 CloudWatch 콘솔을 엽니다.
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 AWSXRayRecorder가 현재 세그먼트 또는 하위 세그먼트 [개체](#)에 대한 참조를 유지하지 않습니다. 새 스레드에서 인스트루먼트된 클라이언트를 사용하는 경우 SDK가 존재하지 않는 세그먼트에 쓰려고 시도하여 오류가 발생합니다.

### [SegmentNotFoundException](#)

개발 중에 예외가 발생하지 않도록 하려면 대신 오류를 기록하도록 [ContextMissingStrategy](#) 지시하는 레코더를 구성할 수 있습니다. 를 사용하여 코드로 [SetContextMissingStrategy](#) 전략을 구성하거나 [환경 변수](#) 또는 [시스템 속성](#)을 사용하여 동등한 옵션을 구성할 수 있습니다.

오류를 처리하는 하나의 방법은 스레드를 시작할 때 [beginSegment](#)를 호출하고 스레드를 닫을 때 [endSegment](#)를 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](#) 단원을 참조하세요.

## 비동기 프로그래밍과 함께 X-Ray 사용하기

Java용 X-Ray SDK는 다음과 같은 비동기 자바 프로그램에서 사용할 수 있습니다.

[SegmentContextExecutors](#) 는 Executor 인터페이스를 SegmentContextExecutor 구현합니다. 즉, a의 모든 비동기 작업에 이 인터페이스를 전달할 수 있습니다. [CompletableFuture](#) 이렇게 하면 모든 비동기 작업이 해당 컨텍스트에서 올바른 세그먼트를 사용하여 실행될 수 있습니다.

Example 예제 App.java: 전달 대상 SegmentContextExecutor CompletableFuture

```
DynamoDbAsyncClient client = DynamoDbAsyncClient.create();

AWSXRay.beginSegment();

// ...

client.getItem(request).thenComposeAsync(response -> {
    // If we did not provide the segment context executor, this request would not be
    // traced correctly.
    return client.getItem(request2);
}, SegmentContextExecutors.newSegmentContextExecutor());
```

## 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 실행 중인 애플리케이션을 비침습적으로 계측할 수 있는 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 추적하기](#)을 참조하세요.

Example src/main/java/myapp/.java WebConfig - 스프링

```
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 [자카르타](#)를 사용합니다. 이 새로운 네임스페이스를 지원하기 위해 X-Ray는 자체 자카르타 네임스페이스에 있는 병렬 클래스 집합을 만들었습니다.

필터 클래스의 경우 javax를 jakarta로 바꾸십시오. 세그먼트 네이밍 전략을 구성할 때는 다음 예시와 같이 네이밍 전략 클래스 이름 앞에 jakarta를 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 `Pointcut`에 대해 호스트의 역할을 합니다. `@XRayEnabled` 주석이 추가된 클래스 중 트레이스할 클래스를 지정하여 `Pointcut`을 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 `AbstractXRayInterceptor`에서 확장하는 것을 고려하세요.

예

다음 코드는 추상 클래스 `BaseAbstractXRayInterceptor`를 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;
```

```

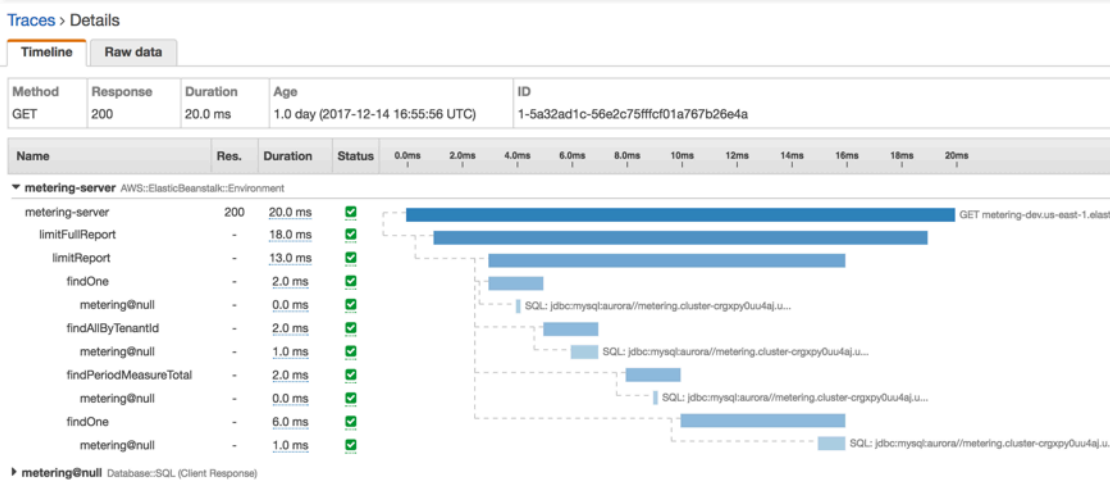
@Autowired
public MyServiceImpl(MyEntityRepository myEntityRepository) {
    this.myEntityRepository = myEntityRepository;
}

@Transactional(readOnly = true)
public List<MyEntity> getMyEntities(){
    try(Stream<MyEntity> entityStream = this.myEntityRepository.streamAll()){

        return entityStream.sorted().collect(Collectors.toList());
    }
}
}

```

애플리케이션을 올바르게 구성했다면 다음 콘솔 스크린샷과 같이 컨트롤러에서 서비스 호출에 이르기까지 애플리케이션의 전체 호출 스택이 보일 것입니다.



## 다음과 같이 애플리케이션을 계측하십시오. Node.js

Node.js 애플리케이션을 계측하여 트레이스를 X-Ray로 전송하는 방법에는 두 가지가 있습니다.

- [AWS 배포판 대상 OpenTelemetry JavaScript — Collector](#)용 AWS 배포판을 통해 Amazon CloudWatch AWS X-Ray, Amazon OpenSearch Service를 비롯한 여러 AWS 모니터링 솔루션으로 상호 관련된 지표와 추적을 전송할 수 있는 오픈 소스 라이브러리 세트를 제공하는 [AWS 배포판입니다](#). OpenTelemetry
- [AWS X-Ray Node.js 용 SDK](#) — X-Ray [데몬](#)을 통해 트레이스를 생성하고 X-Ray로 전송하기 위한 라이브러리 세트입니다.

자세한 내용은 [AWS 배포판용 OpenTelemetry SDK와 X-Ray SDK 중 선택을\(를\)](#) 참조하세요.

## AWS Distro for OpenTelemetry JavaScript

OpenTelemetry JavaScript 용 AWS 배포판 (ADOT)을 사용하면 애플리케이션을 한 번 계측하여 상호 관련된 지표와 추적을 Amazon CloudWatch, AWS X-Ray, Amazon OpenSearch Service를 포함한 여러 AWS 모니터링 솔루션으로 전송할 수 있습니다. OpenTelemetry용 AWS 배포판과 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 위한 AWS 배포판입니다.

시작하려면 [OpenTelemetry JavaScript용 AWS 배포판 설명서](#)를 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 다른 AWS 서비스 버전과 함께 OpenTelemetry용 AWS 배포판을 사용하는 방법에 대한 자세한 내용은 [OpenTelemetry용 AWS 배포판](#) 또는 [OpenTelemetry용 AWS 배포판 설명서](#)를 참조하십시오.

언어 지원 및 사용법에 대한 자세한 내용은 [Github의 AWS 관찰성](#)을 참조하세요.

## AWS Node.js 전용 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. 추적 데이터에는 애플리케이션이 제공하는 수신 HTTP 요청과 애플리케이션이 AWS SDK 또는 HTTP 클라이언트를 사용하여 다운스트림 서비스에 보내는 호출에 대한 정보가 포함됩니다.

### Note

Node.js 용 X-Ray SDK는 Node.js 버전 14.x 이상에서 지원되는 오픈 소스 프로젝트입니다. [github.com/aws/](https://github.com/aws/)에서 프로젝트를 팔로우하고 이슈와 풀 리퀘스트를 제출할 수 있습니다 [GitHub. aws-xray-sdk-node](https://github.com/aws-aws-xray-sdk-node)

Express를 사용한다면, 애플리케이션 서버에서 [SDK를 미들웨어로 추가](#)하여 수신 요청 트레이스를 시작합니다. 미들웨어는 각 트레이스 요청에 대한 [세그먼트](#)를 생성하고, 응답이 전송되면 세그먼트를 완

료합니다. 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 헤더](#)를 읽고 샘플링된 요청을 자동으로 추적합니다. 그 밖의 함수의 경우 수신 요청을 샘플링 및 추적하도록 [Lambda를 구성](#)합니다. 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 정보는 [AWS Lambda](#) 그리고 [AWS X-Ray](#)을 참조하세요.

다음으로 Node.js 용 X-Ray SDK를 사용하여 [Node.js 클라이언트용 AWS SDK를 계측하십시오](#). JavaScript 인스트루먼트 클라이언트를 사용하여 다운스트림 AWS 서비스 또는 리소스에 전화를 걸 때마다 SDK는 해당 호출에 대한 정보를 하위 세그먼트에 기록합니다. AWS 서비스 또한 서비스 내에서 액세스하는 리소스는 추적 맵에 다운스트림 노드로 표시되어 개별 연결의 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.js 용 X-Ray SDK를 구성하여](#) 샘플링 동작을 조정하거나 애플리케이션이 실행되는 AWS 컴퓨팅 리소스에 대한 정보를 기록할 수 있습니다.

요청에 대한 추가 정보와 애플리케이션이 [주석 및 메타데이터](#)에서 하는 작업을 기록합니다. 주석은 [필터 표현식](#)과 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 세그먼트](#)의 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.js용 AWS X-Ray X-Ray SDK API Reference](#)를 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.js 용 X-Ray SDK를 사용하려면 Node.js 형식의 최소 AWS SDK 버전이 필요합니다. JavaScript

- `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.json에 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 문서](#)를 참조하십시오.

## Node.js 샘플

Node.js AWS X-Ray SDK를 사용하여 Node.js 애플리케이션을 통해 전달되는 end-to-end 요청을 파악할 수 있습니다.

- [Node.js 샘플 애플리케이션이](#) 켜져 있습니다. GitHub

## 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

plugins을 사용하여 애플리케이션을 호스팅하는 서비스에 대한 정보를 기록할 수 있습니다.

### 플러그인

- Amazon EC2 — 인스턴스 ID, 가용 영역 및 CloudWatch 로그 그룹을 EC2Plugin 추가합니다.
- Elastic Beanstalk – ElasticBeanstalkPlugin이 환경 이름, 버전 레이블 및 배포 ID를 추가합니다.
- Amazon ECS — ECSPPlugin이 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 > EC2와 같은 해결 순서를 사용하여 출처를 결정합니다.

### 샘플링 규칙

SDK는 X-Ray 콘솔에서 정의하는 샘플링 규칙을 사용하여 기록할 요청을 결정합니다. 기본 규칙은 매 초 첫 번째 요청을 추적하고, 모든 서비스에서 추가 요청의 5%를 X-Ray로 추적 전송합니다. [X-Ray 콘솔에서 추가 규칙을 생성](#)하여 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 X-Ray에 도달하여 샘플링 규칙을 가져올 수 없는 경우 1초가 시작될 때 수신되는 첫 번째 요청의 기본 로컬 규칙과 호스트당 추가 요청의 5%로 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.

아니요 AWS Lambda, 샘플링 속도는 수정할 수 없습니다. 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 `disableCentralizedSampling`를 호출합니다.

```
AWSXRay.middleware.disableCentralizedSampling()
```

## 로깅

SDK의 출력을 기록하려면 `AWSXRay.setLogger(logger)`를 호출합니다. 여기서 `logger`는 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 변수](#)를 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, `Winston`과 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 `setLogger`를 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 변수](#)를 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_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_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 대기](#)하도록 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_MODE`가 `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.js 용 X-Ray SDK를 사용하여 Express 및 Restify 애플리케이션이 Amazon EC2 또는 Amazon ECS의 EC2 인스턴스에서 제공하는 수신 HTTP 요청을 추적할 수 있습니다. AWS Elastic Beanstalk

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 정보는 [AWS Lambda](#) 그리고 [AWS X-Ray](#)을 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](#)
- [Restify를 사용하여 수신 요청 추적](#)
- [세그먼트 이름 지정 전략 구성](#)

### 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 서비스 이름을 사용하여 애플리케이션을 식별하고 애플리케이션이 사용하는 다른 애플리케이션, 데이터베이스, 외부 API 및 AWS 리소스와 구별합니다. X-Ray SDK는 수신 요청에 대한 세그먼트를 생성할 때 해당 세그먼트의 [이름 필드](#)에 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 변수](#)를 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.enableDynamicNaming`을 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.js 용 X-Ray AWS SDK를 사용하여 SDK 호출 추적하기

[애플리케이션이 데이터를 저장하거나, 대기열에 기록하거나, 알림을 보내기 위해 를 호출하는 경우, Node.js X-Ray SDK는 호출을 하위 세그먼트로 다운스트림에서 추적합니다. AWS 서비스 추적되고 AWS 서비스해당 서비스 내에서 액세스하는 리소스 \(예: Amazon S3 버킷 또는 Amazon SQS 대기열\)는 X-Ray 콘솔의 추적 맵에 다운스트림 노드로 표시됩니다.](#)

[V2 또는 V3를 통해 생성한 인스트루먼트 AWS SDK 클라이언트.AWS SDK for JavaScriptAWS SDK for JavaScript](#) 각 AWS SDK 버전은 SDK 클라이언트를 AWS 계측하기 위한 다양한 방법을 제공합니다.

### Note

현재 Node.js AWS X-Ray SDK는 AWS SDK for JavaScript V3 클라이언트를 계측할 때 V2 클라이언트를 계측하는 경우에 비해 더 적은 세그먼트 정보를 반환합니다. 예를 들어, DynamoDB에 대한 직접 호출을 나타내는 하위 세그먼트는 테이블 이름을 반환하지 않습니다. 트레이스에 이 세그먼트 정보가 필요한 경우 V2를 사용해 보세요. AWS SDK for JavaScript

## AWS SDK for JavaScript V2

`aws-sdkrequire` 명령문을 호출에 포함시켜 모든 AWS SDK V2 클라이언트를 계측할 `AWSXRay.captureAWS` 수 있습니다.

Example app.js - AWS SDK 구성

```
const AWS = AWSXRay.captureAWS(require('aws-sdk'));
```

개별 클라이언트를 계측하려면 AWS SDK 클라이언트를 호출로 래핑하십시오.

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 captureAWSClient를 함께 사용하지 마십시오. 이렇게 하면 하위 세그먼트가 중복됩니다.

[ECMAScript 모듈](#) (ESM) [TypeScript](#)과 함께 사용하여 JavaScript 코드를 로드하려면 다음 예제를 사용하여 라이브러리를 가져오십시오.

Example app.js - SDK 인스트루멘테이션 AWS

```
import * as AWS from 'aws-sdk';
import * as AWSXRay from 'aws-xray-sdk';
```

ESM으로 모든 AWS 클라이언트를 계측하려면 다음 코드를 사용하십시오.

Example app.js - AWS SDK 인스트루멘테이션

```
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 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 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": "UBQNS05AEM8T4FDA4RQDEB940VTDRVV4K4HIRGVJF66Q9ASUAAJG",
  }
}
```

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 코드가 필요한 모듈만 로드합니다. 이 때문에 V3에서는 이 메서드를 지원하지 않으므로 모든 AWS SDK 클라이언트를 계측할 수는 없습니다.

captureAWS

ECMAScript 모듈 (ESM) TypeScript 과 함께 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';
```

메서드를 AWS 사용하여 각 SDK 클라이언트를 계측합니다. `AWSXRay.captureAWSSv3Client` 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.captureAWSSv3Client(new DynamoDBClient({ region:
"region" }));
```

AWS SDK for JavaScript V3를 사용하는 경우 테이블 이름, 버킷 및 키 이름 또는 대기열 이름과 같은 메타데이터가 현재 반환되지 않으므로 V2를 사용하여 AWS SDK 클라이언트를 계측할 때처럼 추적 맵에 각 명명된 리소스에 대한 개별 노드가 포함되지 않습니다. AWS SDK for JavaScript

Example V3를 사용할 때 DynamoDB를 호출하여 항목을 저장하기 위한 서브세그먼트 AWS SDK for JavaScript

```
{
  "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": "UBQNS05AEM8T4FDA4RQDEB940VTDRVV4K4HIRGVJF66Q9ASUAAJG",
  }
}
```

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\(\) API](#)를 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 http를 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 의 SQL 쿼리 섹션을 참조하십시오. [X-Ray 세그먼트 문서](#)

## Node.js용 X-Ray SDK를 사용하여 사용자 지정 하위 세그먼트 생성하기

하위 세그먼트는 추적의 [세그먼트](#)를 확장하여 요청을 처리하기 위해 완료된 작업에 대한 세부 정보를 표시합니다. 계속되는 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 `send`라는 사용자 지정 하위 세그먼트를 생성합니다. `captureAsyncFunc`는 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 Trace 맵에 단일 `AWS::Lambda::Function` 노드를 생성하는 기본 하위 세그먼트를 만들려면 파사드 세그먼트를 호출하고 용도를 변경하십시오. 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

주석과 메타데이터를 사용하여 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 표현식](#)에서 사용하기 위해 인덱싱됩니다. 주석은 콘솔의 트레이스를 그룹화할 때 사용할 데이터를 기록하거나 [GetTraceSummaries](#) API를 호출할 때 사용하십시오.

메타데이터는 객체 및 목록을 포함한 모든 유형의 값을 가질 수 있는 키-값 페어이지만, 필터 표현식에 사용할 수 있도록 인덱싱되지는 않습니다. 트레이스에 저장하고 싶지만 검색에는 사용하지 않을 추가 데이터는 메타데이터를 사용하여 기록하십시오.

세그먼트에는 주석과 메타데이터 외에 [사용자 ID 문자열](#)도 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. 밑줄 기호 (\_) 이외의 공백이나 기호는 사용할 수 없습니다.
- 값 — X-Ray 주석의 값은 최대 1,000개의 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 `addAnnotation`을 호출합니다.

```
document.addAnnotation("mykey", "my value");
```

SDK는 세그먼트 문서의 `annotations` 객체에 주석을 키-값 페어로 기록합니다. 같은 키로 `addAnnotation`을 두 번 호출하면 같은 세그먼트나 하위 세그먼트에 기록했던 값을 덮어씁니다.

특정 값을 포함한 주석이 있는 트레이스를 찾으려면 `annotations.key` 키워드를 [필터 표현식](#)에 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 addMetadata를 호출합니다.

```
document.addMetadata("my key", "my value", "my namespace");
```

또는

키와 값만 이용해 addMetadata를 호출합니다.

```
document.addMetadata("my key", "my value");
```

네임스페이스를 지정하지 않으면, SDK는 default를 사용합니다. 같은 키로 addMetadata을 두 번 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 없으므로 이 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 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 표현식](#)에 적용하십시오.

## 다음과 같이 애플리케이션을 계측하십시오 Python

트레이스를 X-Ray로 전송하도록 Python 애플리케이션을 계측하는 방법에는 두 가지가 있습니다.

- [AWS 배포판 대상 OpenTelemetry Python — Collector용](#) AWS 배포판을 통해 Amazon CloudWatch, AWS X-Ray, Amazon OpenSearch Service를 비롯한 여러 AWS 모니터링 솔루션으로 상호 관련된 지표와 추적을 전송할 수 있는 오픈 소스 라이브러리 세트를 제공하는 [AWS 배포판입니다](#). OpenTelemetry

- [AWS X-Ray SDK for Python](#) — X-Ray [데몬](#)을 통해 트레이스를 생성하고 X-Ray로 전송하기 위한 라이브러리 세트입니다.

자세한 내용은 [AWS 배포판용 OpenTelemetry SDK와 X-Ray SDK 중 선택](#)(을)를 참조하세요.

## AWS 배포판 OpenTelemetry Python

AWS Distrofor OpenTelemetry (ADOT) 를 사용하면 애플리케이션을 한 번 계측하고 Amazon Python CloudWatch, Amazon Service를 포함한 여러 AWS 모니터링 솔루션에 상관 관계가 있는 AWS X-Ray 지표와 추적을 전송할 수 있습니다. OpenSearch ADOT와 함께 X-Ray를 사용하려면 두 가지 구성 요소가 필요합니다. 하나는 X-Ray와 함께 사용할 수 있는 OpenTelemetry SDK이고 다른 하나는 X-Ray와 함께 사용할 수 있는 OpenTelemetryCollector용AWS 배포판입니다. PythonADOT에는 자동 계측 지원이 포함되어 있어 애플리케이션에서 코드 변경 없이 트레이스를 전송할 수 있습니다.

[시작하려면 배포판에서 설명서를 참조하십시오.AWS OpenTelemetry Python](#)

OpenTelemetry [함께 AWS X-Ray 배포판을 사용하거나 다른 AWS 서비스용으로 사용하는 방법](#)에 대한 자세한 내용은 [설명서용 배포판 OpenTelemetry 또는 AWS 설명서용 배포판을 참조하십시오.AWS OpenTelemetry](#)

[언어 지원 및 사용에 대한 자세한 내용은 Observability on을 참조하십시오AWS . GitHub](#)

## AWS X-Ray SDK를 위한 Python

Python용 X-Ray SDK는 트레이스 데이터를 생성하고 X-Ray 데몬으로 전송하기 위한 클래스와 메서드를 제공하는 Python 웹 애플리케이션용 라이브러리입니다. 추적 데이터에는 응용 프로그램에서 제공하는 수신 HTTP 요청 및 응용 프로그램이 AWS SDK, HTTP 클라이언트 또는 SQL 데이터베이스 커넥터를 사용하여 다운스트림 서비스에 보내는 호출에 대한 정보가 포함됩니다. 또한 수동으로 세그먼트를 생성하고 디버그 정보를 주석 및 메타데이터에 추가할 수도 있습니다.

pip로 SDK를 다운로드할 수 있습니다.

```
$ pip install aws-xray-sdk
```

### Note

Python용 X-Ray SDK는 오픈 소스 프로젝트입니다. [github.com/aws/](https://github.com/aws/)에서 프로젝트를 팔로우하고 이슈와 풀 리퀘스트를 제출할 수 있습니다 [GitHub. aws-xray-sdk-python](https://github.com/aws-xray-sdk-python)

Django 또는 Flask를 사용하는 경우 [SDK 미들웨어를 애플리케이션에 추가하는 것부터](#) 시작하여 수신 요청을 추적합니다. 미들웨어는 각 트race 요청에 대한 [세그먼트](#)를 생성하고, 응답이 전송되면 세그먼트를 완료합니다. 세그먼트가 열려 있는 동안에는 SDK 클라이언트의 메서드를 이용해 정보를 세그먼트에 추가하고 하위 세그먼트를 만들어 다운스트림 호출을 트race할 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 헤더](#)를 읽고 샘플링된 요청을 자동으로 추적합니다. 그 밖의 함수의 경우 수신 요청을 샘플링 및 추적하도록 [Lambda를 구성](#)합니다. 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 정보는 [AWS Lambda](#) 그리고 [AWS X-Ray](#)을 참조하세요.

Lambda에서 계측된 예제 Python 함수는 [작업자](#) 참조하십시오.

다음으로 Python용 X-Ray SDK를 사용하여 [애플리케이션의 라이브러리를 패치함](#)으로써 다운스트림 호출을 구성합니다. SDK는 다음 라이브러리를 지원합니다.

지원되는 라이브러리

- [botocore](#), [boto3](#) — 인스트루먼트 클라이언트. AWS SDK for Python (Boto)
- [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](#)— PyMy MySQL 및 MariaDB를 위한 인스트루먼트 SQL 기반 클라이언트

애플리케이션이 SQL 데이터베이스 또는 기타 HTTP 서비스를 호출할 AWS때마다 SDK는 호출에 대한 정보를 하위 세그먼트에 기록합니다. AWS 서비스 또한 서비스 내에서 액세스하는 리소스는 추적 맵에 다운스트림 노드로 나타나므로 개별 연결의 오류 및 제한 문제를 식별하는 데 도움이 됩니다.

SDK를 사용하기 시작한 후에, [레코더와 미들웨어를 구성](#)하여 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 메타데이터](#)에서 하는 작업을 기록합니다. 주석은 [필터 표현식](#)과 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 세그먼트](#)의 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 설명서는 API용 [AWS X-Ray SDK](#) 참조를 참조하십시오. Python

## 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
- 열거형 — 0.4.7 이상, 버전 3.4.0 이상용 Python
- jsonpickle – 1.0.0 이상
- setuptools – 40.6.3 이상
- wrapt – 1.11.0 이상

## 종속성 관리

Python용 X-Ray SDK는 pip에서 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](#)
- [샘플링 규칙](#)

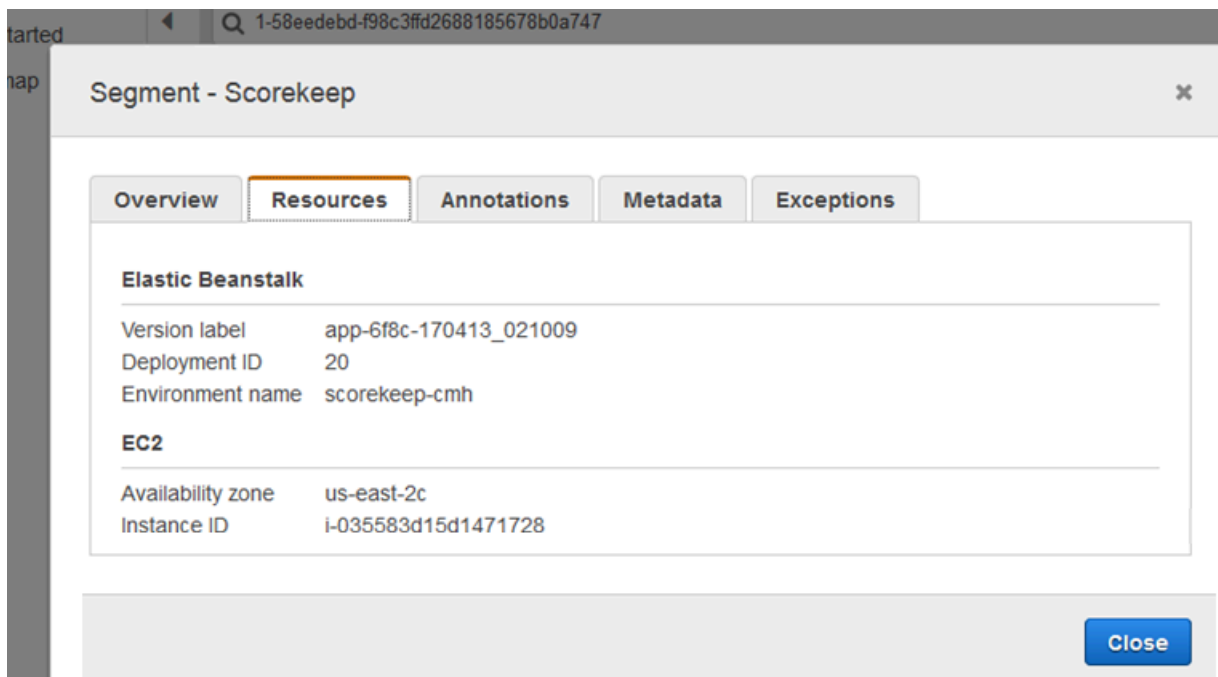
- [로깅](#)
- [코드의 레코더 구성](#)
- [Django의 레코더 구성](#)
- [환경 변수](#)

## 서비스 플러그인

plugins을 사용하여 애플리케이션을 호스팅하는 서비스에 대한 정보를 기록할 수 있습니다.

## 플러그인

- Amazon EC2 — 인스턴스 ID, 가용 영역 및 CloudWatch 로그 그룹을 EC2Plugin 추가합니다.
- Elastic Beanstalk – ElasticBeanstalkPlugin이 환경 이름, 버전 레이블 및 배포 ID를 추가합니다.
- Amazon ECS — ECSPlugin이 컨테이너 ID를 추가합니다.



플러그인을 사용하려면 xray\_recorder에서 configure를 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

plugins는 튜플로 전달되므로 단일 플러그인을 지정할 때는 반드시 후행 , 를 포함해야 합니다. 예제: plugins = ('EC2Plugin',)

또한 코드에 설정된 값보다 우선하는 [환경 변수](#)를 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 > EC2와 같은 해결 순서를 사용하여 출처를 결정합니다.

### 샘플링 규칙

SDK는 X-Ray 콘솔에서 정의하는 샘플링 규칙을 사용하여 기록할 요청을 결정합니다. 기본 규칙은 매 초 첫 번째 요청을 추적하고, 모든 서비스에서 추가 요청의 5%를 X-Ray로 추적 전송합니다. [X-Ray 콘솔에서 추가 규칙을 생성](#)하여 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 X-Ray에 도달하여 샘플링 규칙을 가져올 수 없는 경우 1초가 시작될 때 수신되는 첫 번째 요청의 기본 로컬 규칙과 호스트당 추가 요청의 5%로 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.

아니오 AWS Lambda, 샘플링 속도는 수정할 수 없습니다. 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 디렉터리이거나 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 비활성화하고 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 setLevel을 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 생성](#)할 때 미완료 하위 세그먼트와 같은 문제를 식별할 수 있습니다.

## 코드의 레코더 구성

xray\_recorder에 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\_ERROR로 설정합니다.
- daemon\_address – X-Ray 대몬(daemon) 리스너의 호스트와 포트를 설정합니다.
- service – SDK가 세그먼트에 사용하는 서비스 이름을 설정합니다.
- plugins – 애플리케이션 AWS 리소스에 대한 정보를 기록합니다.
- sampling – 샘플링을 비활성화하려면 False로 설정합니다.
- sampling\_rules – [샘플링 규칙](#)이 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`— 애플리케이션 AWS 리소스에 대한 정보를 기록합니다.
- `SAMPLING` – 샘플링을 비활성화하려면 `False`로 설정합니다.
- `SAMPLING_RULES` – [샘플링 규칙](#)이 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_RECORDER`인 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', 'ECSPPlugin'),
}
```

```
'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` – `false`로 설정하면 SDK가 비활성화됩니다. 이 환경 변수가 `false`로 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 `127.0.0.1:2000`을 사용합니다. [다른 포트에서 수신 대기](#)하도록 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_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.

## 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 샘플링되는 각 요청마다 세그먼트 하나를 생성합니다. 자세한 정보는 [AWS Lambda](#) 그리고 [AWS X-Ray](#) 섹션을 참조하세요.

Lambda로 계측된 Python 함수의 예는 [작업자](#)를 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.

## 섹션

- [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 구성](#)할 수 있습니다.

#### Note

plugins는 튜플로 전달되므로 단일 플러그인을 지정할 때는 반드시 후행 , 를 포함해야 합니다. 예: plugins = ('EC2Plugin',)

애플리케이션(Flask)에 미들웨어 추가

Flask 애플리케이션을 구성하려면 먼저 xray\_recorder에서 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 구성](#)할 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 구성](#)할 수 있습니다.

### 수동으로 Python 코드 구성

Django 또는 Flask를 사용하지 않는 경우 세그먼트를 수동으로 만들 수 있습니다. 각 수신 요청에 대해 세그먼트를 만들거나 패치된 HTTP 또는 AWS SDK 클라이언트 주변에 세그먼트를 만들어서 레코더에서 하위 세그먼트를 추가할 수 있는 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-Ray는 서비스 이름을 사용하여 사용자의 애플리케이션을 식별하고, 이를 다른 애플리케이션, 데이터베이스, 외부 API, 사용자의 애플리케이션에서 사용하는 AWS 리소스와 구별합니다. X-Ray SDK는 수신 요청에 대한 세그먼트를 생성할 때 해당 세그먼트의 [이름 필드](#)에 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 섹션](#)에 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 변수](#)를 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 구성합니다.
- [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 사용](#)을 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 `patch`를 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

- http-lib - [http-lib](#) 및 [http.client](#)
- mysql - [mysql-connector-python](#)

## 비동기 작업에 대한 컨텍스트 추적

asyncio 통합 라이브러리의 경우 또는 [비동기 함수에 대한 하위 세그먼트를 생성](#)하려면 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 추적하기

[애플리케이션이 데이터를 저장하거나, 대기열에 기록하거나, 알림을 보내기 위해 를 호출하는 경우, X-Ray SDK for Python은 호출을 하위 세그먼트로 다운스트림에서 추적합니다.](#) AWS 서비스 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 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": "UBQNS05AEM8T4FDA4RQDEB940VTDREV4K4HIRGVJF66Q9ASUAAJG",
```

```
}
}
```

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 패치](#)합니다. requests 또는 Python의 내장 HTTP 클라이언트를 사용하는 경우 더 이상 수행할 작업이 없습니다. aiohttp의 경우 [비동기 컨텍스트](#)를 사용하여 레코더를 구성합니다.

aiohttp 3의 클라이언트 API를 사용하면, SDK에서 제공하는 추적 구성의 인스턴스로 ClientSession의 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 [세그먼트](#)를 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

주석과 메타데이터를 사용하여 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 표현식](#)에서 사용하기 위해 인덱싱됩니다. 주석은 콘솔의 트레이스를 그룹화할 때 사용할 데이터를 기록하거나 [GetTraceSummaries](#) API를 호출할 때 사용하십시오.

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 문자열](#)도 기록할 수 있습니다. 사용자 ID는 세그먼트의 별도 필드에 기록되면 검색용으로 인덱스되지 않습니다.

### Sections

- [Python용 X-Ray SDK로 주석 기록하기](#)
- [Python용 X-Ray SDK로 메타데이터 기록하기](#)
- [Python용 X-Ray SDK로 사용자 ID 기록하기](#)

## 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. 밑줄 기호 (\_) 이외의 공백이나 기호는 사용할 수 없습니다.
- 값 — X-Ray 주석의 값은 최대 1,000개의 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()
```

또는

```
from aws_xray_sdk.core import xray_recorder
...
document = xray_recorder.current_subsegment()
```

2. 문자열 키, 부울, 숫자 또는 문자열 값으로 `put_annotation`을 호출합니다.

```
document.put_annotation("mykey", "my value");
```

또는 `put_annotation`에서 `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_annotation`을 두 번 호출하면 같은 세그먼트나 하위 세그먼트에 기록했던 값을 덮어씁니다.

특정 값을 포함한 주석이 있는 트레이스를 찾으려면 `annotations.key` 키워드를 [필터 표현식](#)에 사용하십시오.

## Python용 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_recorder`에서 현재 세그먼트나 하위 세그먼트의 참조를 가져오십시오.

```
from aws_xray_sdk.core import xray_recorder
...
document = xray_recorder.current_segment()
```

또는

```
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_metadata`를 호출합니다.

```
document.put_metadata("my key", "my value", "my namespace");
```

또는

키와 값만 이용해 `put_metadata`를 호출합니다.

```
document.put_metadata("my key", "my value");
```

또는 `put_metadata`에서 `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 `default`를 사용합니다. 같은 키로 `put_metadata`을 두 번 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 호출합니다.

```
document.set_user("U12345");
```

컨트롤러에서 `set_user`를 호출하면 애플리케이션이 요청을 처리하는 순간부터 사용자 ID를 기록할 수 있습니다.

사용자 ID의 트레이스를 찾으려면, `user` 키워드를 [필터 표현식](#)에 적용하십시오.

## 서버리스 환경에 배포된 웹 프레임워크 구성

Python용 AWS X-Ray SDK는 서버리스 애플리케이션에 배포된 계층 웹 프레임워크를 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 배포되는 Flask 또는 Django와 같은 웹 AWS X-Ray 프레임워크에서 자동으로 계층하는 방법을 보여줍니다. 애플리케이션의 X-Ray 계층을 사용하면 Amazon API Gateway에서 시작하여 AWS Lambda 함수를 통해 이루어진 모든 다운스트림 호출과 애플리케이션이 보내는 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.

### 필수 조건

- [Zappa](#)
- [Python](#) – 버전 2.7 또는 3.6.
- [AWS CLI](#)— 계정을 AWS CLI 사용하여 구성되어 있고 AWS 리전 애플리케이션을 배포할 계정이 어 디인지 확인하십시오.
- [Pip](#)
- [Virtualenv](#)

### 단계 1: 환경 조성

이 단계에서는 virtualenv를 사용하여 애플리케이션을 호스팅할 가상 환경을 만듭니다.

1. 를 사용하여 응용 프로그램의 디렉토리를 생성합니다. AWS CLI그런 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` 디렉토리에 애플리케이션 코드를 추가합니다. 이 예제에서는 Flask의 [Hello World](#) 예제를 빌드합니다.

`serverless_application` 디렉토리에서 `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` 디렉토리에서 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.

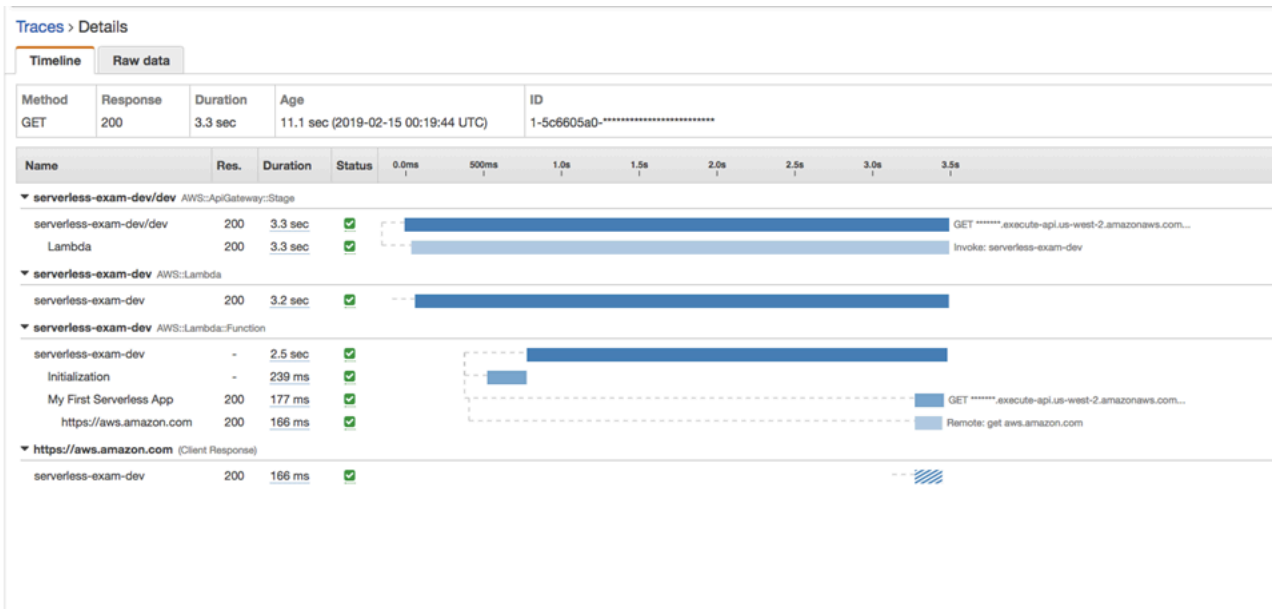
1. <https://console.aws.amazon.com/apigateway/> 에서 AWS Management Console 로그인하고 API Gateway 콘솔을 엽니다.
2. 새로 생성한 API를 찾습니다. 이 URL은 serverless-exam-dev과 유사해야 합니다.
3. 단계를 선택합니다.
4. 배포 단계의 이름을 선택합니다. 기본값은 dev입니다.
5. Logs/Tracing(로그/추적) 탭에서 Enable X-Ray Tracing(X-Ray 추적 활성화) 확인란을 선택합니다.
6. 변경 사항 저장(Save Changes)을 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 보기](#)를 참조하십시오.

1. <https://console.aws.amazon.com/xray/home> 에서 AWS Management Console 로그인하고 X-Ray 콘솔을 엽니다.
2. API 게이트웨이, Lambda 함수 및 Lambda 컨테이너에 의해 생성된 세그먼트를 봅니다.
3. Lambda 함수 세그먼트 아래에서 하위 세그먼트 My First Serverless App을 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 추가하십시오. [서버리스 온 AWS](#)에서 서버리스 컴퓨팅 옵션에 대해 자세히 알아보십시오.

## 다음과 같이 애플리케이션을 계측하십시오 .NET

트레이스를 X-Ray로 전송하도록 .NET 애플리케이션을 계측하는 방법에는 두 가지가 있습니다.

- [AWS 배포판 대상 OpenTelemetry .NET — Collector](#)용 AWS 배포판을 통해 Amazon CloudWatch AWS X-Ray, Amazon OpenSearch Service를 비롯한 여러 AWS 모니터링 솔루션으로 상호 관련된 지표와 추적을 전송할 수 있는 오픈 소스 라이브러리 세트를 제공하는 [AWS 배포판입니다](#). OpenTelemetry
- [AWS X-Ray SDK for .NET](#) — X-Ray [데몬](#)을 통해 트레이스를 생성하고 X-Ray로 전송하기 위한 라이브러리 세트입니다.

자세한 내용은 [AWS 배포판용 OpenTelemetry SDK와 X-Ray SDK 중 선택](#)(를) 참조하세요.

## AWS 배포판 용 OpenTelemetry .NET

AWS 배포판을 사용하면 애플리케이션을 한 번 계측하고 Amazon OpenTelemetry .NET CloudWatch AWS X-Ray, Amazon OpenSearch Service를 비롯한 여러 AWS 모니터링 솔루션으로 상호 관련된 지표와 추적을 전송할 수 있습니다. AWS Distro와 함께 X-Ray를 OpenTelemetry 사용하려면 두 가지 구성 요소가 필요합니다. 하나는 X-Ray와 함께 사용할 수 있는 OpenTelemetry SDK이고 다른 하나는 X-Ray와 함께 사용할 수 있는 OpenTelemetry Collector용AWS Distro입니다.

[시작하려면 배포판에서 설명서를 참조하십시오.AWS OpenTelemetry .NET](#)

OpenTelemetry [함께 AWSAWS X-Ray 배포판을 사용하거나 다른 AWS 서비스용으로 사용하는 방법에 대한 자세한 내용은 설명서용 배포판 OpenTelemetry 또는 AWS 설명서용 배포판을 참조하십시오.AWS OpenTelemetry](#)

[언어 지원 및 사용에 대한 자세한 내용은 Observability on을 참조하십시오AWS . GitHub](#)

## 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 Lambda이 라이브러리는 트레이스 데이터를 생성하여 [X-Ray 대](#)

[몬\(daemon\)](#)으로 전송하기 위한 클래스 및 메서드를 제공합니다. 여기에는 응용 프로그램에서 제공하는 수신 요청, 응용 프로그램이 다운스트림 AWS 서비스, HTTP 웹 API 및 SQL 데이터베이스에 대해 수행하는 호출에 대한 정보가 포함됩니다.

### Note

.NET용 X-Ray SDK는 오픈 소스 프로젝트입니다. [github.com/aws/](https://github.com/aws/)에서 프로젝트를 팔로우하고 이슈와 풀 리퀘스트를 제출할 수 있습니다 [GitHub. aws-xray-sdk-dotnet](https://github.com/aws-xray-sdk-dotnet)

웹 애플리케이션의 경우 [메시지 핸들러](#)를 웹 구성에 추가하여 수신 요청을 트race하는 것부터 시작합니다. 메시지 핸들러는 각 트race 요청에 대한 [세그먼트](#)를 생성하고, 응답이 전송되면 세그먼트를 완료합니다. 세그먼트가 열려 있는 동안에는 SDK 클라이언트의 메서드를 이용해 정보를 세그먼트에 추가하고 하위 세그먼트를 만들어 다운스트림 호출을 트race할 수 있습니다. 또한 SDK는 세그먼트가 열려 있는 동안 애플리케이션에서 발생하는 예외를 자동으로 기록합니다.

계측되는 애플리케이션 또는 서비스에 의해 호출되는 Lambda 함수의 경우, Lambda는 [추적 헤더](#)를 읽고 샘플링된 요청을 자동으로 추적합니다. 그 밖의 함수의 경우 수신 요청을 샘플링 및 추적하도록 [Lambda를 구성](#)합니다. 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](#)를 참조하세요.

그런 다음, X-Ray SDK for .NET를 사용하여 [AWS SDK for .NET 클라이언트를 계측](#) 합니다. 인스트루먼트된 클라이언트를 사용하여 다운스트림 AWS 서비스 또는 리소스에 전화를 걸 때마다 SDK는 해당 호출에 대한 정보를 하위 세그먼트에 기록합니다. AWS 서비스 내에서 액세스하는 서비스와 리소스는 추적 맵에 다운스트림 노드로 나타나므로 개별 연결의 오류 및 제한 문제를 식별하는 데 도움이 됩니다.

또한 .NET용 X-Ray SDK는 [HTTP 웹 API](#) 및 [SQL 데이터베이스](#)에 대한 다운스트림 호출의 계측도 제공합니다. System.Net.HttpWebRequest에 대한 GetResponseTraced 확장 메서드는 발신 HTTP 호출을 트race합니다. .NET용 X-Ray SDK 버전의 SqlCommand를 사용하여 SQL 쿼리를 계측할 수 있습니다.

SDK를 사용하기 시작한 후에, [레코더와 메시지 핸들러를 구성](#)하여 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 메타데이터](#)에서 하는 작업을 기록합니다. 주석은 [필터 표현식](#)과 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-Ray 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 세그먼트](#)의 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 레퍼런스](#)
- [AWS X-Ray .NET용 SDK 코어 API 레퍼런스](#)

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 SDK를 사용하려면 .NET 프레임워크 4.5 이상 버전이 필요합니다. AWS SDK for .NET

.NET Core 애플리케이션 및 함수의 경우, SDK에는 .NET Core 2.0 이상이 필요합니다.



## 애플리케이션에 .NET용 X-Ray SDK 추가하기

.NET용 X-Ray SDK를 애플리케이션에 NuGet 추가하는 데 사용합니다.

비주얼 스튜디오에서 패키지 관리자를 NuGet 사용하여 .NET용 X-Ray SDK를 설치하려면

1. [도구], [NuGet 패키지 관리자], [솔루션용 NuGet 패키지 관리] 를 선택합니다.
2. AWSXRayRecorder를 찾습니다.
3. 패키지를 선택하고 Install(설치)을 선택합니다.

### 종속성 관리

.NET용 X-Ray SDK는 [Nuget](#)에서 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 해결](#)에 대한 Microsoft의 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 구성](#)할 때 UseXRay 메서드에 구성 객체를 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 구성](#)을 참조하세요.

## Sections

- [플러그인](#)
- [샘플링 규칙](#)
- [로깅 \(.NET\)](#)

- [로깅\(.NET Core\)](#)
- [환경 변수](#)

## 플러그인

플러그인을 사용하여 애플리케이션을 호스팅하는 서비스에 대한 데이터를 추가합니다.

## 플러그인

- Amazon EC2 — 인스턴스 ID, 가용 영역 및 CloudWatch 로그 그룹을 EC2Plugin 추가합니다.
- Elastic Beanstalk – ElasticBeanstalkPlugin이 환경 이름, 버전 레이블 및 배포 ID를 추가합니다.
- Amazon ECS — ECSPlugin이 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 매 초 첫 번째 요청을 추적하고, 모든 서비스에서 추가 요청의 5%를 X-Ray로 추적 전송합니다. [X-Ray 콘솔에서 추가 규칙을 생성](#)하여 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 X-Ray에 도달하여 샘플링 규칙을 가져올 수 없는 경우 1초가 시작될 때 수신되는 첫 번째 요청의 기본 로컬 규칙과 호스트당 추가 요청의 5%로 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.

네 AWS Lambda, 샘플링 속도는 수정할 수 없습니다. 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 .NET](#)와 동일한 로깅 메커니즘을 사용합니다. AWS SDK for .NET 출력을 기록하도록 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 구성](#)을 참조하십시오.

## 로깅(.NET Core)

.NET용 X-Ray SDK는 [AWS SDK for .NET](#)와 동일한 로깅 옵션을 사용합니다. .NET Core 응용 프로그램의 로깅을 구성하려면 로깅 옵션을 `AWSXRayRecorder.RegisterLogger` 메서드에 전달하십시오.

예를 들어 `log4net`을 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>
</log4net>
```

```

</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.org](http://logging.apache.org)의 [구성](#)을 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 전략](#)에 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 127.0.0.1:2000을 사용합니다. [다른 포트에서 수신 대기](#)하도록 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_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.

## .NET용 X-Ray SDK로 수신 HTTP 요청 계측하기

X-Ray SDK를 사용하여 애플리케이션이 Amazon EC2, AWS Elastic Beanstalk 또는 Amazon ECS의 EC2 인스턴스에서 처리하는 수신 HTTP 요청을 트race할 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 샘플링되는 각 요청마다 세그먼트 하나를 생성합니다. 자세한 정보는 [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.

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-agent입니다.
- 콘텐츠 길이 — 응답의 content-length입니다.

### 섹션

- [수신 HTTP 요청 구성\(.NET\)](#)
- [수신 HTTP 요청 구성\(.NET Core\)](#)
- [세그먼트 이름 지정 전략 구성](#)

### 수신 HTTP 요청 구성(.NET)

애플리케이션에 의해 저장된 요청을 구성하려면 RegisterXRay 파일의 Init 메서드에서 global.asax를 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 객체](#)를 2차 인수로 사용할 수도 있습니다.

```
app.UseXRay("MyApp", configuration);

```

### 세그먼트 이름 지정 전략 구성

AWS X-Ray는 서비스 이름을 사용하여 사용자의 애플리케이션을 식별하고, 이를 다른 애플리케이션, 데이터베이스, 외부 API, 사용자의 애플리케이션에서 사용하는 AWS 리소스와 구별합니다. X-Ray SDK는 수신 요청에 대한 세그먼트를 생성할 때 해당 세그먼트의 [이름 필드](#)에 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 단원](#)과 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 변수](#)를 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 [DynamicSegmentNamingStrategy](#)를 생성하고 이를 RegisterXRay 메서드에 전달합니다.

```
AWSXRayASPNET.RegisterXRay(this, new DynamicSegmentNamingStrategy("MyApp",
    "*.example.com"));
```

.NET용 X-Ray AWS SDK를 사용하여 SDK 호출을 추적합니다.

[애플리케이션이 AWS 서비스 호출하여 데이터를 저장하거나, 대기열에 기록하거나, 알림을 보내는 경우, .NET용 X-Ray SDK는 하위 세그먼트의 다운스트림 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.

클라이언트를 RegisterXRayForAllServices 생성하기 전에 전화를 걸어 모든 AWS SDK for .NET 클라이언트를 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
    {
        AWS SDK Handler.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 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": "UBQNS05AEM8T4FDA4RQDEB940VTDTRVV4K4HIRGVJF66Q9ASUAAJG",
  }
}
```

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.httpclient](#)를 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.SqlCommand용 래퍼 클래스 TraceableSqlCommand를 제공합니다. TraceableSqlCommand 클래스를 사용하여 SQL 명령을 초기화할 수 있습니다.

동기 및 비동기 메서드를 사용하여 SQL 쿼리 추적

다음 예제에서는 TraceableSqlCommand를 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.CommandText는 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 true로 설정합니다.
- TraceableSqlCommand 인스턴스를 collectSqlQueries 파라미터를 true로 설정하여 인스턴스 내의 호출을 수집합니다.

## 글로벌 CollectSqlQueries 속성 활성화

다음 예제에서는 .NET 및 .NET Core에 대해 CollectSqlQueries 속성을 활성화하는 방법을 보여줍니다.

### .NET

.NET의 애플리케이션에 대한 글로벌 구성에서 CollectSqlQueries 속성을 true로 설정하려면 다음과 같이 App.config 또는 Web.config 파일의 appsettings를 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 [세그먼트](#)를 확장하여 요청을 처리하기 위해 완료된 작업에 대한 세부 정보를 표시합니다. 계속되는 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

주석과 메타데이터를 사용하여 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 표현식](#)에서 사용하기 위해 인덱싱됩니다. 주석은 콘솔의 트레이스를 그룹화할 때 사용할 데이터를 기록하거나 [GetTraceSummaries](#) API를 호출할 때 사용하십시오.

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. 밑줄 기호 (\_) 이외의 공백이나 기호는 사용할 수 없습니다.
- 값 — X-Ray 주석의 값은 최대 1,000개의 유니코드 문자를 포함할 수 있습니다.
- 주석 수 — 트레이스당 최대 50개의 주석을 사용할 수 있습니다.

## 함수 외부에 주석을 기록하려면 AWS Lambda

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");
```

## 함수 내부에 주석을 기록하려면 AWS Lambda

Lambda 함수 내의 세그먼트와 하위 세그먼트 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. 세그먼트나 하위 세그먼트에 주석을 추가합니다.
3. 세그먼트나 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 주석을 세그먼트 문서의 annotations 개체에 키-값 쌍으로 기록합니다. 동일한 키로 addAnnotation 작업을 두 번 호출하면 동일한 세그먼트 또는 하위 세그먼트에 이전에 기록된 값을 덮어씁니다.

특정 값의 주석이 있는 트레이스를 찾으려면 필터 표현식에서 annotations.*key* 키워드를 사용하십시오. 자세한 정보는 [필터 표현식 사용](#)을 참조하세요.

## .NET용 X-Ray SDK로 메타데이터 기록하기

메타데이터를 사용하면 검색에서 사용하기 위해 인덱싱할 필요가 없는 세그먼트나 하위 세그먼트에 대한 정보를 기록할 수 있습니다. 메타데이터 값은 문자열, 숫자, 부울 또는 JSON 객체 또는 배열로 직렬화할 수 있는 기타 객체일 수 있습니다.

### 메타데이터 기록 방법

1. 다음 코드 예제와 AWSXRayRecorder 같이 의 인스턴스를 가져옵니다.

```
using Amazon.XRay.Recorder.Core;
...
AWSXRayRecorder recorder = AWSXRayRecorder.Instance;
```

2. 다음 코드 AddMetadata 예제와 같이 문자열 네임스페이스, 문자열 키, 개체 값을 사용하여 호출합니다.

```
recorder.AddMetadata("my namespace", "my key", "my value");
```

다음 코드 예제와 같이 키와 값 쌍만 사용하여 AddMetadata 작업을 호출할 수도 있습니다.

```
recorder.AddMetadata("my key", "my value");
```

네임스페이스 값을 지정하지 않으면 X-Ray SDK가 사용합니다. default 동일한 키로 AddMetadata 작업을 두 번 호출하면 동일한 세그먼트 또는 하위 세그먼트에서 이전에 기록된 값을 덮어씁니다.

## Ruby로 애플리케이션을 계속하세요

Ruby 애플리케이션을 계속하여 트레이스를 X-Ray로 전송하는 방법에는 두 가지가 있습니다.

- [AWS OpenTelemetry Ruby용 배포판](#) — Collector용 AWS 배포판을 통해 Amazon CloudWatch, AWS X-Ray Amazon OpenSearch Service를 비롯한 여러 AWS 모니터링 솔루션으로 상호 관

련된 지표와 추적을 전송할 수 있는 오픈 소스 라이브러리 세트를 제공하는 [AWS 배포판입니다](#).

OpenTelemetry

- [AWS X-Ray Ruby용 SDK — X-Ray 데몬을 통해 트레이스를 생성하고 X-Ray로 전송하기 위한 라이브러리 세트입니다](#).

자세한 내용은 [AWS 배포판용 OpenTelemetry SDK와 X-Ray SDK 중 선택을\(를\)](#) 참조하세요.

## AWS Distro for OpenTelemetry Ruby

Ruby용 OpenTelemetry AWS 배포판 (ADOT)을 사용하면 애플리케이션을 한 번 계측하여 상호 관련된 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 위한 AWS 배포판입니다.

시작하려면 [Ruby용 OpenTelemetry AWS 배포판 설명서](#)를 참조하십시오.

AWS X-Ray 및 다른 AWS 서비스 버전과 함께 OpenTelemetry AWS 배포판을 사용하는 방법에 대한 자세한 내용은 [OpenTelemetry용 AWS 배포판](#) 또는 [OpenTelemetry용 AWS 배포판 설명서](#)를 참조하십시오.

언어 지원 및 사용법에 대한 자세한 내용은 [Github의 AWS 관찰성](#)을 참조하세요.

## AWS X-Ray 루비용 SDK

X-Ray SDK는 트레이스 데이터를 생성하고 X-Ray 데몬(daemon)에 보내기 위한 클래스 및 메서드를 제공하는 Ruby 웹 애플리케이션용 라이브러리 집합입니다. 추적 데이터에는 애플리케이션이 제공하는 수신 HTTP 요청과 애플리케이션이 AWS SDK, HTTP 클라이언트 또는 활성 레코드 클라이언트를 사용하여 다운스트림 서비스에 보내는 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 추가](#)하세요. 요청 필터는 [세그먼트](#)를 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.

다음으로, 관련 라이브러리를 패치하도록 [레코더를 구성하여](#) X-Ray SDK를 사용하여 HTTP 및 SQL 클라이언트를 계측합니다. AWS SDK for Ruby 인스턴트루먼트된 클라이언트를 사용하여 다운스트림 AWS 서비스 또는 리소스를 호출할 때마다 SDK는 해당 호출에 대한 정보를 하위 세그먼트에 기록합니다. AWS 서비스 또한 서비스 내에서 액세스하는 리소스는 추적 맵에 다운스트림 노드로 표시되어 개별 연결의 오류 및 제한 문제를 식별하는 데 도움이 됩니다.

SDK를 이용하게 되면 [레코더를 구성](#)하여 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 메타데이터](#)에서 하는 작업을 기록합니다. 주석은 [필터 표현식](#)과 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-Ray 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 세그먼트](#)의 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 Reference](#)를 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

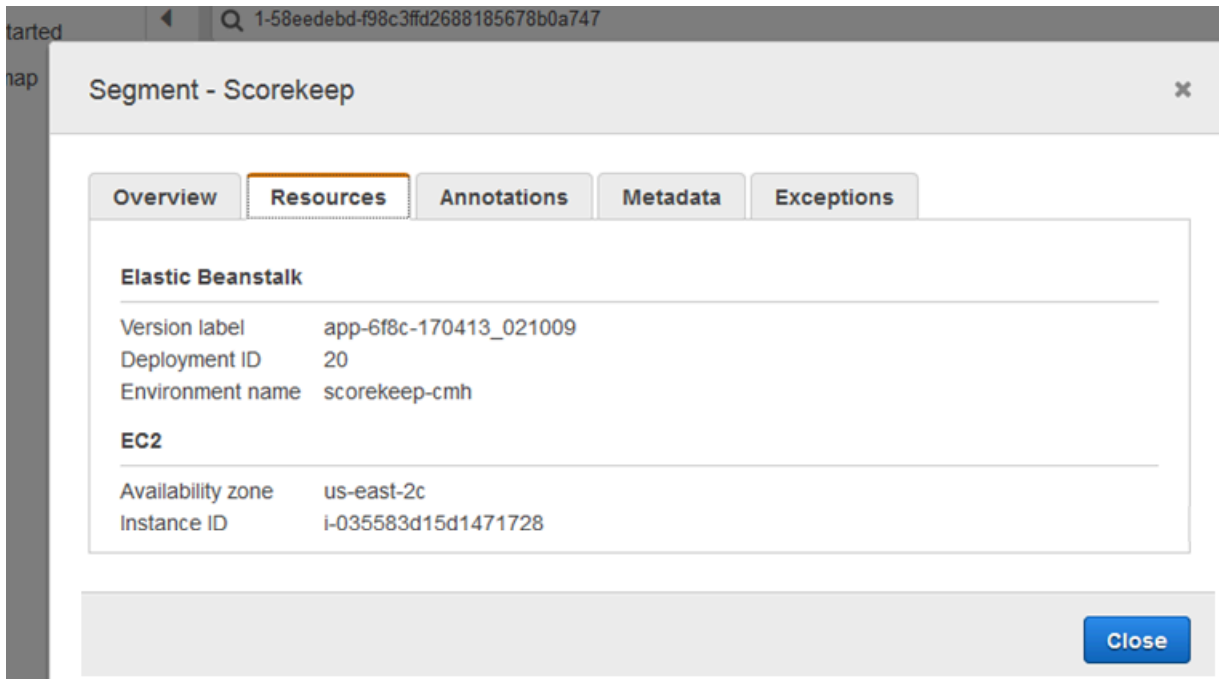
- [서비스 플러그인](#)
- [샘플링 규칙](#)
- [로깅](#)
- [코드의 레코더 구성](#)
- [Rails를 사용한 레코더 구성](#)
- [환경 변수](#)

### 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 변수](#)를 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 > EC2와 같은 해결 순서를 사용하여 출처를 결정합니다.

#### 샘플링 규칙

SDK는 X-Ray 콘솔에서 정의하는 샘플링 규칙을 사용하여 기록할 요청을 결정합니다. 기본 규칙은 매 초 첫 번째 요청을 추적하고, 모든 서비스에서 추가 요청의 5%를 X-Ray로 추적 전송합니다. [X-Ray 콘솔에서 추가 규칙을 생성](#)하여 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 X-Ray에 도달하여 샘플링 규칙을 가져올 수 없는 경우 1초가 시작될 때 수신되는 첫 번째 요청의 기본 로컬 규칙과 호스트당 추가 요청의 5%로 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 LocalSampler를 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 비활성화하고 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 [로거](#)를 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 생성](#)할 때 미완료 하위 세그먼트와 같은 문제를 식별할 수 있습니다.

## 코드의 레코더 구성

XRay.recorder에 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 지정](#)을 사용하려면 도메인 이름 패턴을 설정합니다.
- `plugins` – [플러그인](#)을 사용하여 애플리케이션의 AWS 리소스에 대한 정보를 기록합니다.
- `sampling` – 샘플링을 비활성화하려면 `false`로 설정합니다.
- `sampling_rules` – [샘플링 규칙](#)이 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 전략](#)에 설정한 서비스 이름을 재정의합니다.
- **AWS\_XRAY\_DAEMON\_ADDRESS** – X-Ray 대몬(daemon) 리스너의 호스트와 포트를 설정합니다. 기본적으로 SDK는 127.0.0.1:2000에 트레이스 데이터를 보냅니다. [다른 포트에서 수신 대기](#)하도록 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 또는 Amazon ECS의 EC2 인스턴스에서 처리하는 수신 HTTP 요청을 추적할 수 있습니다. AWS Elastic Beanstalk

Rails를 사용하는 경우 Rails 미들웨어를 사용하여 수신 HTTP 요청을 구성합니다. 미들웨어를 애플리케이션에 추가하고 세그먼트 이름을 구성하면 Ruby용 X-Ray SDK는 샘플링된 각 요청에 대해 세그먼트를 생성합니다. 추가 구성에 의해 생성된 모든 세그먼트는 HTTP 요청 및 응답에 대한 정보를 제공하



는 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 [railtie](#)를 포함하도록 `gemfile`을 업데이트합니다.

#### Example Gemfile - rails

```
gem 'aws-xray-sdk', require: ['aws-xray-sdk/facets/rails/railtie']
```

또한 미들웨어를 사용하려면 추적 [맵의 애플리케이션을 나타내는 이름으로 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 만들거나 패치가 적용된 HTTP 또는 AWS SDK 클라이언트를 중심으로 세그먼트를 생성하여 레코더가 하위 세그먼트를 추가할 수 있는 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 서비스 이름을 사용하여 애플리케이션을 식별하고 애플리케이션이 사용하는 다른 애플리케이션, 데이터베이스, 외부 API 및 AWS 리소스와 구별합니다. X-Ray SDK는 수신 요청에 대한 세그먼트를 생성할 때 해당 세그먼트의 [이름 필드](#)에 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 섹션](#)에 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 변수](#)를 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 구성합니다.

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 추적하기

[애플리케이션이 데이터를 저장하거나, 대기열에 기록하거나, 알림을 보내기 위해 호출하면 Ruby용 X-Ray는 호출을 하위 세그먼트로 다운스트림으로 추적합니다. AWS 서비스](#) 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-Ray는 라이브러리를 [패치할 때](#) 모든 SDK 클라이언트를 자동으로 AWS 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 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": "UBQNS05AEM8T4FDA4RQDEB940VTDRVV4K4HIRGVJF66Q9ASUAAJG",
  }
}
```

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 [세그먼트](#)를 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

주석과 메타데이터를 사용하여 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 표현식](#)에서 사용하기 위해 인덱싱됩니다. 주석은 콘솔의 트레이스를 그룹화할 때 사용할 데이터를 기록하거나 [GetTraceSummaries](#) API를 호출할 때 사용하십시오.

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 문자열](#)도 기록할 수 있습니다. 사용자 ID는 세그먼트의 별도 필드에 기록되면 검색용으로 인덱스되지 않습니다.

## Sections

- [Ruby용 X-Ray SDK로 주석 기록하기](#)
- [Ruby용 X-Ray SDK로 메타데이터 기록하기](#)
- [Ruby용 X-Ray SDK로 사용자 ID 기록하기](#)

## 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. 밑줄 기호 (\_) 이외의 공백이나 기호는 사용할 수 없습니다.
- 값 — X-Ray 주석의 값은 최대 1,000개의 유니코드 문자를 포함할 수 있습니다.
- 주석 수 — 트레이스당 최대 50개의 주석을 사용할 수 있습니다.

### 주석 기록 방법

1. xray\_recorder에서 현재 세그먼트나 하위 세그먼트의 참조를 가져오십시오.

```
require 'aws-xray-sdk'
...
document = XRay.recorder.current_segment
```

또는

```
require 'aws-xray-sdk'
...
document = XRay.recorder.current_subsegment
```

2. 해시 값을 사용해 update를 호출합니다.

```
my_annotations = { id: 12345 }
document.annotations.update my_annotations
```

SDK는 세그먼트 문서의 annotations 객체에 주석을 키-값 페어로 기록합니다. 같은 키로 add\_annotations을 두 번 호출하면 같은 세그먼트나 하위 세그먼트에 기록했던 값을 덮어씁니다.

특정 값을 포함한 주석이 있는 트레이스를 찾으려면 annotations.key 키워드를 [필터 표현식](#)에 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
```

또는

```
require 'aws-xray-sdk'
...
document = XRay.recorder.current_subsegment
```

2. 문자열 키, 부울, 숫자, 문자열 또는 객체 값 및 문자열 네임스페이스와 함께 metadata를 호출합니다.

```
my_metadata = {
  my_namespace: {
    key: 'value'
  }
}
subsegment.metadata my_metadata
```



같은 키로 metadata을 두 번 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 표현식](#)에 적용하십시오.

## 다른 AWS X-Ray 사람과 통합 AWS 서비스

많은 업체에서 샘플링 및 수신 요청에 헤더 추가, X-Ray 데몬 실행, 추적 데이터를 X-Ray로 자동 전송하는 등 다양한 수준의 X-Ray 통합 기능을 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 데몬(daemon)을 실행하여 X-Ray SDK로부터 세그먼트를 수신합니다.

### Note

X-Ray SDK에는 추가 통합을 위한 플러그인이 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 통합된 몇 AWS 서비스 가지 예입니다.

- [AWS 배포판 OpenTelemetry \(ADOT\)](#) — 엔지니어는 ADOT를 사용하여 애플리케이션을 한 번 계측하여 상호 관련된 지표와 추적을 Amazon CloudWatch, AWS X-Ray Amazon 서비스 및 Prometheus용 Amazon Managed Service를 포함한 여러 모니터링 AWS 솔루션으로 보낼 수 있습니다. OpenSearch
- [AWS Lambda](#)— 모든 런타임에서 들어오는 요청을 능동적 및 수동적으로 계측합니다. AWS Lambda 추적 맵에 두 개의 노드를 추가합니다. 하나는 AWS Lambda 서비스용이고 다른 하나는 기능용입니다. 계측을 활성화하면 Java 및 Node.js 런타임에서 X-Ray 데몬을 실행하여 X-Ray SDK와 함께 사용할 AWS Lambda 수도 있습니다.
- [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 데몬(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.
- [아마존 EventBridge](#) — 패시브 인스트루멘테이션. 이벤트를 게시하는 서비스가 X-Ray SDK로 인스트루먼트되면 이벤트 대상은 추적 헤더를 수신하고 원래 추적 ID를 계속 전파할 수 있습니다. EventBridge
- [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.

다음 주제 중에서 선택하여 전체 통합 세트를 살펴보세요. AWS 서비스

## 주제

- [AWS Distro for OpenTelemetry 및 AWS X-Ray](#)
- [Amazon API Gateway 액티브 트레이싱 지원 대상 AWS X-Ray](#)
- [아마존 EC2 및 AWS App Mesh](#)
- [AWS App Runner 및 X-Ray](#)
- [AWS AppSync 및 AWS X-Ray](#)
- [를 사용하여 X-Ray API 호출 로깅 AWS CloudTrail](#)
- [CloudWatch X-Ray와의 통합](#)
- [AWS Config를 사용하여 X-Ray 암호화 구성 변경 추적하기](#)
- [Amazon Elastic Compute Cloud와 AWS X-Ray](#)
- [AWS Elastic Beanstalk 및 AWS X-Ray](#)
- [Elastic Load Balancing 및 AWS X-Ray](#)
- [아마존 EventBridge 및 AWS X-Ray](#)

- [AWS Lambda 그리고 AWS X-Ray](#)
- [아마존 SNS 및 AWS X-Ray](#)
- [AWS Step Functions 및 AWS X-Ray](#)
- [아마존 SQS 및 AWS X-Ray](#)
- [아마존 S3와 AWS X-Ray](#)

## AWS Distro for OpenTelemetry 및 AWS X-Ray

OpenTelemetry용 AWS 배포판 (ADOT) 을 사용하여 지표와 추적을 수집하여 AWS X-Ray와 Amazon CloudWatch, Amazon OpenSearch Service, Prometheus용 Amazon Managed Service 등의 기타 모니터링 솔루션으로 전송할 수 있습니다.

### AWS Distro for OpenTelemetry

오픈 텔레메트리용 AWS 배포판 (ADOT) 은 클라우드 네이티브 컴퓨팅 재단 (CNCF) 의 오픈 텔레메트리 프로젝트를 기반으로 하는 AWS 배포판입니다. 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 애플리케이션을 한 번 계측하여 상호 관련된 지표와 추적을 Amazon CloudWatch, AWS X-Ray, Amazon OpenSearch Service, Prometheus용 Amazon Managed Service를 포함한 여러 AWS 모니터링 솔루션으로 전송할 수 있습니다.

ADOT는 점점 더 많은 AWS 서비스와 통합되어 추적 및 메트릭을 X-Ray와 같은 모니터링 솔루션으로 전송하는 작업을 간소화합니다. ADOT와 통합된 서비스의 몇 가지 예는 다음과 같습니다:

- AWS Lambda— ADOT용 AWS 관리형 Lambda 계층은 Lambda 함수를 자동으로 계측하고, OpenTelemetry를 바로 사용할 수 있는 구성 및 설정이 간편한 계층에 AWS Lambda 및 X-Ray와 함께 패키징함으로써 플러그 앤 플레이 사용자 환경을 제공합니다. 사용자는 코드를 변경하지 않고도 Lambda 함수에 대해 OpenTelemetry를 활성화 및 비활성화할 수 있습니다. 자세한 내용은 [OpenTelemetry Lambda용 AWS 배포판](#)을 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 수집](#)을 참조하세요.

- AWS 앱 러너 — App Runner는 OpenTelemetry용 AWS 배포판(ADOT)을 사용하여 X-Ray에 추적을 전송할 수 있도록 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-Ray](#)를 참조하세요.

추가 AWS 서비스와의 통합을 포함하여 OpenTelemetry용 AWS 배포판에 대한 자세한 내용은 [OpenTelemetry용 AWS 배포판 설명서](#)를 참조하세요.

OpenTelemetry용 AWS Distro 및 X-Ray를 사용하여 애플리케이션을 계측하는 방법에 대한 자세한 내용은 [OpenTelemetry용 AWS Distro를 사용하여 애플리케이션 계측하기](#)를 참조하세요.

## Amazon API Gateway 액티브 트레이싱 지원 대상 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. Amazon API Gateway와 함께 X-Ray를 사용할 수 있는 모든 곳에서 X-Ray를 사용할 수 있습니다. 자세한 내용을 알아보려면 Amazon API Gateway 개발자 안내서의 [AWS X-Ray를 사용한 API 게이트웨이 API 실행 추적](#)을 참조하세요.

### Note

X-Ray는 API 게이트웨이를 통한 REST API에 대한 추적만 지원합니다.

Amazon API Gateway는 에 대한 [AWS X-Ray능동적 추적](#) 지원을 제공합니다. 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 설명은 [샘플링 규칙을 구성합니다](#). 섹션을 참조하세요.

### Note

API Gateway [HTTP 통합](#)을 통해 REST API를 추적할 때 각 세그먼트의 서비스 이름은 API Gateway에서 HTTP 통합 엔드포인트로 향하는 요청 URL 경로로 설정되므로 각 고유 URL 경로에 대한 X-Ray 추적 맵에 서비스 노드가 생성됩니다. URL 경로가 많으면 추적 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 헤더](#)가 아직 없는 수신 HTTP 요청에 추적 헤더를 추가합니다.

```
X-Amzn-Trace-Id: Root=1-5759e988-bd862e3fe1be46a994272793
```

### X-Ray 트레이스 ID 형식

X-Ray trace\_id는 하이픈으로 구분된 3개의 숫자로 구성됩니다. 예를 들어 1-58406520-a006649127e371903a2de979입니다. 여기에는 다음이 포함됩니다.

- 버전 번호, 즉 1.
- 원래 요청 시간을 Unix 시대 시간으로 나타낸 8자리 16진수

예를 들어 2016년 12월 1일 오전 10시 (PST) 기준 에포크 타임은 초 또는 16진수로 표시됩니다.  
1480615200 58406520

- 24자리 16진수 트레이스를 위한 전 세계적으로 고유한 96비트 식별자입니다.

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 계

측하면 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 인스트루먼트하는 것과 동일한 방식으로 클라이언트 빌더에서 요청 핸들러를 추가하여 SDK 클라이언트를 계속할 수 있습니다. AWS 자세한 내용은 [Java용 X-Ray AWS SDK를 사용하여 SDK 호출 추적하기](#) 섹션을 참조하세요.

## 아마존 EC2 및 AWS App Mesh

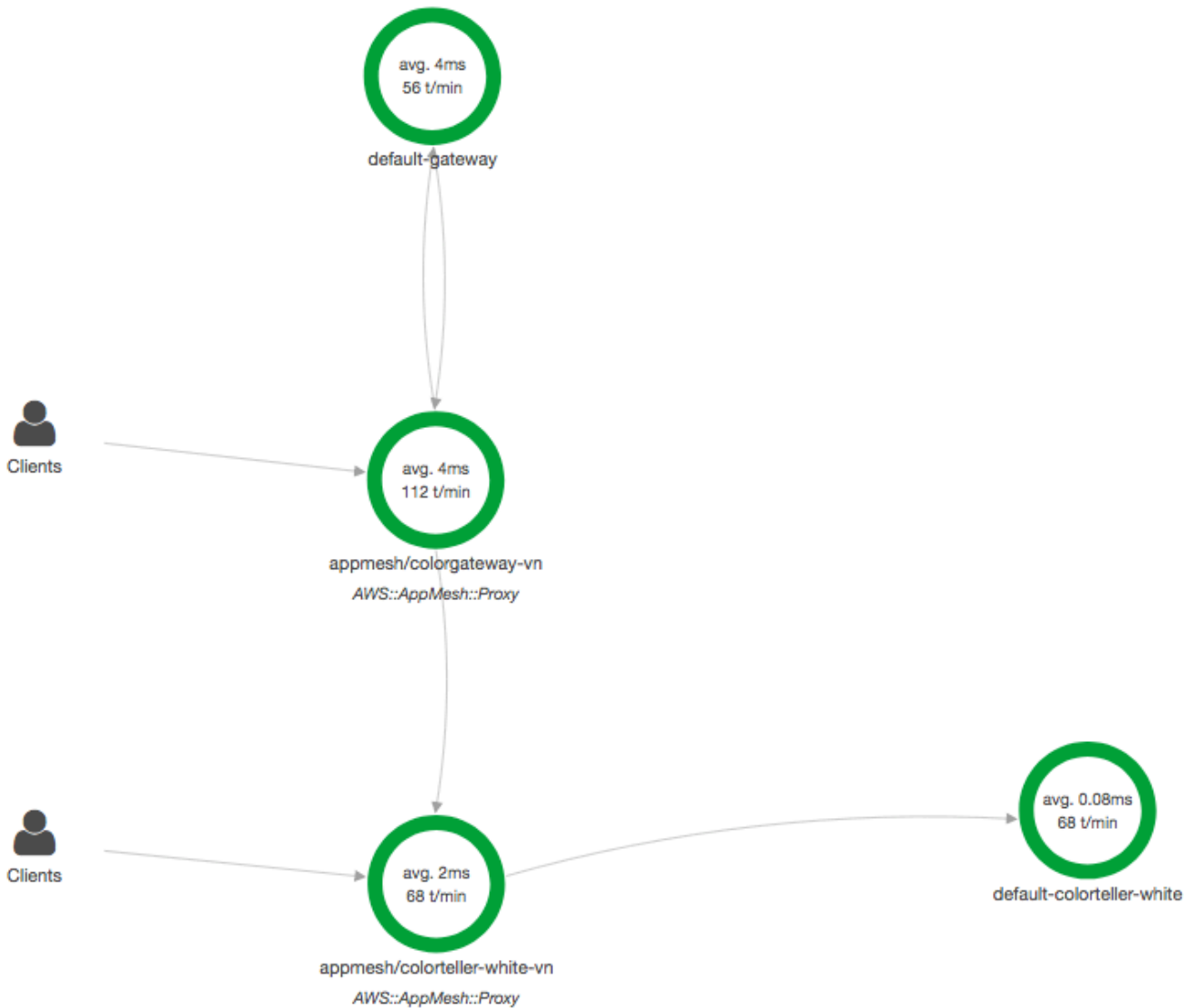
AWS X-Ray 와 통합하여 마이크로서비스용 [AWS App Mesh](#) Envoy 프록시를 관리합니다. App Mesh는 동일한 작업 또는 포드의 컨테이너에서 실행 중인 X-Ray 대몬(daemon)에 추적 데이터를 전송하도록 구성할 수 있는 Envoy 버전을 제공합니다. X-Ray는 다음과 같은 App Mesh 호환 서비스를 통한 추적을 지원합니다.

- 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.

## Service map

Enter a service name to find and select the node on map



Envoy 프록시가 X-Ray로 데이터를 전송하도록 구성하려면 컨테이너 정의에서 `ENABLE_ENVOY_XRAY_TRACING` [환경 변수](#)를 설정합니다.

**Note**

Envoy의 App Mesh 버전은 현재 구성된 [샘플링 규칙](#)을 기반으로 트race를 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 이미지](#)를 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 데몬이 포함된 샘플 애플리케이션의 경우 App Mesh Examples 저장소에 샘플을 colorapp 배포하십시오. GitHub](#)

### 자세히 알아보기

- [AWS App Mesh 시작하기](#)

- [Amazon ECS AWS App Mesh 시작하기](#)

## AWS App Runner 및 X-Ray

AWS App Runner는 소스 코드 또는 컨테이너 이미지에서 AWS 클라우드의 확장 가능하고 안전한 웹 애플리케이션에 직접 배포할 수 있는 빠르고 간단하며 비용 효율적인 방법을 제공하는 AWS 서비스입니다. 새로운 기술을 배우거나, 사용할 컴퓨팅 서비스를 결정하거나, AWS 리소스를 프로비저닝하고 구성하는 방법을 익힐 필요가 없습니다. 자세한 내용은 [AWS App Runner란 무엇입니까?](#)를 참조하십시오.

AWS App Runner는 [OpenTelemetry용 AWS 배포판 \(ADOT\)](#) 과 통합하여 X-Ray에 트레이스를 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.

## AWS AppSync 및 AWS X-Ray

AWS AppSync에 대한 요청을 활성화하고 추적할 수 있습니다. 자세한 내용은 [AWS X-Ray로 추적하기](#)를 참조하십시오.

AWS AppSync API에 대해 X-Ray 추적을 활성화하면 계정에 적절한 권한이 있는 AWS Identity and Access Management [서비스 연결 역할](#)이 자동으로 생성됩니다. 이를 통해 AWS AppSync에서 X-Ray로 추적을 안전하게 전송할 수 있습니다.

## 를 사용하여 X-Ray API 호출 로깅 AWS CloudTrail

AWS X-Ray 사용자 [AWS CloudTrail](#), 역할 또는 담당자가 수행한 작업의 기록을 제공하는 서비스와 통합되어 AWS 서비스입니다. CloudTrail X-Ray에 대한 모든 API 호출을 이벤트로 캡처합니다. 캡처된 통화에는 X-Ray 콘솔의 통화와 X-Ray API 작업에 대한 코드 호출이 포함됩니다. 에서 수집한 CloudTrail 정보를 사용하여 X-Ray에 대한 요청, 요청이 이루어진 IP 주소, 요청 시기 및 추가 세부 정보를 확인할 수 있습니다.

모든 이벤트 및 로그 항목에는 요청을 생성한 사용자에 대한 정보가 들어 있습니다. 보안 인증 정보를 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 사용자에게 대한 임시 보안 인증 정보를 사용하여 요청이 생성되었는지 여부.

- 다른 AWS 서비스에서 요청했는지 여부.

CloudTrail 계정을 만들 AWS 계정 때 활성화되며 자동으로 CloudTrail 이벤트 기록에 액세스할 수 있습니다. CloudTrail 이벤트 기록은 지난 90일간의 기록된 관리 이벤트를 보고, 검색하고, 다운로드할 수 있고, 변경할 수 없는 기록을 제공합니다. AWS 리전자세한 내용은 사용 설명서의 [CloudTrail 이벤트 기록 사용](#)을 참조하십시오. AWS CloudTrail 이벤트 기록 조회에는 CloudTrail 요금이 부과되지 않습니다.

AWS 계정 지난 90일 동안 진행 중인 이벤트 기록을 보려면 트레일 또는 [CloudTrail호수](#) 이벤트 데이터 저장소를 생성하세요.

## CloudTrail 트레일

트레일을 사용하면 CloudTrail Amazon S3 버킷으로 로그 파일을 전송할 수 있습니다. 를 사용하여 생성된 모든 트레일은 멀티 AWS Management Console 리전입니다. AWS CLI를 사용하여 단일 리전 또는 다중 리전 추적을 생성할 수 있습니다. 계정의 모든 활동을 기록할 수 있으므로 멀티 리전 트레일을 생성하는 것이 좋습니다 AWS 리전 . 단일 리전 추적을 생성하는 경우 추적의 AWS 리전에 로그인된 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 organization](#)을 참조하세요.

트레일을 CloudTrail 생성하여 진행 중인 관리 이벤트의 사본 하나를 Amazon S3 버킷으로 무료로 전송할 수 있지만 Amazon S3 스토리지 요금이 부과됩니다. CloudTrail 요금에 대한 자세한 내용은 [AWS CloudTrail 요금](#)을 참조하십시오. Amazon S3 요금에 대한 자세한 내용은 [Amazon S3 요금](#)을 참조하세요.

## CloudTrail 레이크 이벤트 데이터 스토어

CloudTrail Lake를 사용하면 이벤트에 대한 SQL 기반 쿼리를 실행할 수 있습니다. CloudTrail [Lake](#)는 [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 선택기](#)를 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 사용 설명서의 Lake [사용](#)을 참조하십시오. AWS CloudTrail AWS CloudTrail

CloudTrail Lake 이벤트 데이터 저장 및 쿼리로 인해 비용이 발생합니다. 이벤트 데이터 스토어를 생성할 때 이벤트 데이터 스토어에 사용할 [요금 옵션](#)을 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 요금](#)을 참조하십시오.

## 주제

- [의 X-Ray 관리 이벤트 CloudTrail](#)
- [의 X-Ray 데이터 이벤트 CloudTrail](#)
- [X-Ray 이벤트 예제](#)

## 의 X-Ray 관리 이벤트 CloudTrail

AWS X-Ray 와 AWS CloudTrail 통합하여 사용자, 역할 또는 AWS 서비스 X-Ray에서 수행한 API 작업을 기록합니다. 를 CloudTrail 사용하여 X-Ray API 요청을 실시간으로 모니터링하고 Amazon S3, Amazon 로그 및 Amazon CloudWatch Events에 CloudWatch 로그를 저장할 수 있습니다. X-Ray는 다음 작업을 CloudTrail 로그 파일에 이벤트로 기록할 수 있도록 지원합니다.

### 지원되는 API 작업

- [PutEncryptionConfig](#)
- [GetEncryptionConfig](#)
- [CreateGroup](#)
- [UpdateGroup](#)
- [DeleteGroup](#)
- [GetGroup](#)
- [GetGroups](#)
- [GetInsight](#)
- [GetInsightEvents](#)
- [GetInsightImpactGraph](#)
- [GetInsightSummaries](#)
- [GetSamplingStatisticSummaries](#)

## 의 X-Ray 데이터 이벤트 CloudTrail

[데이터 이벤트](#)는 리소스에서 또는 리소스에서 수행된 리소스 작업에 대한 정보를 제공합니다 (예: 세그먼트 문서를 X-Ray에 업로드하는 경우). [PutTraceSegments](#)

이를 데이터 영역 작업이라고도 합니다. 데이터 이벤트가 대량 활동인 경우도 있습니다. 기본적으로는 데이터 이벤트를 기록하지 CloudTrail 않습니다. CloudTrail 이벤트 기록에는 데이터 이벤트가 기록되지 않습니다.

데이터 이벤트에는 추가 요금이 적용됩니다. CloudTrail 요금에 대한 자세한 내용은 [AWS CloudTrail 요금](#)을 참조하십시오.

CloudTrail 콘솔 또는 CloudTrail API 작업을 사용하여 X-Ray 리소스 유형에 대한 데이터 이벤트를 기록할 수 있습니다. AWS CLI 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 Interface](#)를 참조하세요.

다음 표에는 데이터 이벤트를 기록할 수 있는 X-Ray 리소스 유형이 나열되어 있습니다. 데이터 이벤트 유형 (콘솔) 열에는 CloudTrail 콘솔의 데이터 이벤트 유형 목록에서 선택할 수 있는 값이 표시됩니다. resources.type 값 열에는 또는 resources.type API를 사용하여 고급 이벤트 선택기를 구성할 때 지정하는 값이 표시됩니다. AWS CLI CloudTrail 데이터 API 로깅 대상 CloudTrail 열에는 해당 리소스 유형에 대해 로깅된 API 호출이 표시됩니다. CloudTrail

데이터 이벤트 유형(콘솔)	resources.type 값	로깅된 데이터 API CloudTrail
X-Ray 트레이스	AWS::XRay::Trace	<ul style="list-style-type: none"> <li>• <a href="#">PutTraceSegments</a></li> <li>• <a href="#">GetTraceSummaries</a></li> <li>• <a href="#">GetTraceGraph</a></li> <li>• <a href="#">GetServiceGraph</a></li> <li>• <a href="#">BatchGetTraces</a></li> <li>• <a href="#">GetTimeSeriesServiceStatistics</a></li> <li>• <a href="#">PutTelemetryRecords</a></li> <li>• <a href="#">GetSamplingTargets</a></li> </ul>

eventName 및 readOnly 필드를 필터링하여 중요한 이벤트만 기록하도록 고급 이벤트 선택기를 구성할 수 있습니다. 하지만 X-Ray 추적에는 ARN이 없으므로 resources.ARN 필드 선택기를 추가하여 이벤트를 선택할 수는 없습니다. 이러한 필드에 대한 자세한 내용은 AWS CloudTrail API 참조의 [AdvancedFieldSelector](#) 섹션을 참조하세요. 다음은 [put-event-selectors](#) AWS CLI 명령을 실행하여 트레이스에 데이터 이벤트를 기록하는 방법의 예입니다. CloudTrail 명령을 실행하거나 트레이스가 생성된 지역을 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 X-Ray `GetEncryptionConfig` 로그 항목의 예입니다 CloudTrail.

#### Example

```

{
  "eventVersion"=>"1.05",
  "userIdentity"=>{
    "type"=>"AssumedRole",
    "principalId"=>"AR0AJVHBZWD3DN6CI2MHH: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"=>"AR0AJVHBZWD3DN6CI2MHH",
        "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 X-Ray PutTraceSegments 데이터 이벤트 로그 항목의 예입니다 CloudTrail.

### Example

```

{
  "eventVersion": "1.09",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AROAWYXPW54Y4NEXAMPLE:i-0dzz2ac111c83zz0z",
    "arn": "arn:aws:sts::012345678910:assumed-role/my-service-role/i-0dzz2ac111c83zz0z",
    "accountId": "012345678910",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AROAWYXPW54Y4NEXAMPLE",
        "arn": "arn:aws:iam::012345678910:role/service-role/my-service-role",
        "accountId": "012345678910",
        "userName": "my-service-role"
      },
      "attributes": {
        "creationDate": "2024-01-22T17:34:11Z",
        "mfaAuthenticated": "false"
      }
    },
    "ec2RoleDelivery": "2.0"
  }
}

```

```

},
"eventTime": "2024-01-22T18:22:05Z",
"eventSource": "xray.amazonaws.com",
"eventName": "PutTraceSegments",
"awsRegion": "us-west-2",
"sourceIPAddress": "198.51.100.0",
"userAgent": "aws-sdk-ruby3/3.190.0 md/internal ua/2.0 api/xray#1.0.0 os/linux md/
x86_64 lang/ruby#2.7.8 md/2.7.8 cfg/retry-mode#legacy",
"requestParameters": {
  "traceSegmentDocuments": [
    "trace_id:1-00zzz24z-EXAMPLE4f4e41754c77d0000",
    "trace_id:1-00zzz24z-EXAMPLE4f4e41754c77d0000",
    "trace_id:1-00zzz24z-EXAMPLE4f4e41754c77d0001",
    "trace_id:1-00zzz24z-EXAMPLE4f4e41754c77d0002"
  ]
},
"responseElements": {
  "unprocessedTraceSegments": []
},
"requestID": "5zzzzz64-acbd-46ff-z544-451a3ebcb2f8",
"eventID": "4zz51z7z-77f9-44zz-9bd7-6c8327740f2e",
"readOnly": false,
"resources": [
  {
    "type": "AWS::XRay::Trace"
  }
],
"eventType": "AwsApiCall",
"managementEvent": false,
"recipientAccountId": "012345678910",
"eventCategory": "Data",
"tlsDetails": {
  "tlsVersion": "TLSv1.2",
  "cipherSuite": "ZZZZ-RSA-AAA128-GCM-SHA256",
  "clientProvidedHostHeader": "example.us-west-2.xray.cloudwatch.aws.dev"
}
}

```

## CloudWatch X-Ray와의 통합

AWS X-Ray [CloudWatch 애플리케이션 신호](#), CloudWatch RUM 및 CloudWatch Synthetics와 통합되어 애플리케이션 상태를 보다 쉽게 모니터링할 수 있습니다. 애플리케이션 신호 애플리케이션을 활성화



화하여 서비스, 클라이언트 페이지, Synthetics canaries 및 서비스 종속성의 운영 상태를 모니터링하고 문제를 해결할 수 있습니다.

CloudWatch 메트릭, 로그 및 X-Ray 트레이스를 상호 연관시키는 X-Ray trace 맵은 서비스 end-to-end 뷰를 제공하여 성능 병목 현상을 신속하게 찾아내고 영향을 받는 사용자를 식별하는 데 도움이 됩니다.

CloudWatch RUM을 사용하면 실제 사용자 모니터링을 수행하여 실제 사용자 세션에서 웹 애플리케이션 성능에 대한 클라이언트 측 데이터를 거의 실시간으로 수집하고 볼 수 있습니다. CloudWatch RUM을 사용하면 애플리케이션의 최종 사용자부터 시작하여 다운스트림 관리 서비스에 이르는 요청 경로를 분석하고 디버깅할 수 있습니다. AWS X-Ray AWS 이를 통해 최종 사용자에게 영향을 미치는 지연 시간 추세와 오류를 파악할 수 있습니다.

## 주제

- [CloudWatch 램 및 AWS X-Ray](#)
- [X-Ray를 사용한 CloudWatch 합성 카나리아 디버깅](#)

## CloudWatch 램 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. CloudWatch RUM을 사용하면 애플리케이션의 최종 사용자부터 시작하여 다운스트림 관리 서비스에 이르는 요청 경로를 분석하고 디버그할 수 있습니다. AWS X-Ray AWS 이를 통해 최종 사용자에게 영향을 미치는 지연 시간 추세와 오류를 파악할 수 있습니다.

사용자 세션의 X-Ray 추적 기능을 켜면 CloudWatch RUM은 허용된 HTTP 요청에 X-Ray 추적 헤더를 추가하고 허용된 HTTP 요청에 대해 X-Ray 세그먼트를 기록합니다. 그러면 X-Ray trace 맵을 포함하여 X-Ray와 CloudWatch 콘솔에서 이러한 사용자 세션의 트레이스 및 세그먼트를 볼 수 있습니다.

### Note

CloudWatch RUM은 X-Ray 샘플링 규칙과 통합되지 않습니다. 대신 CloudWatch RUM을 사용하도록 애플리케이션을 설정할 때 샘플링 비율을 선택하십시오. CloudWatch RUM에서 전송된 트레이스는 추가 비용이 발생할 수 있습니다. 자세한 내용은 [AWS X-Ray 요금](#)을 참조하십시오.

기본적으로 CloudWatch RUM에서 전송된 클라이언트 측 트레이스는 서버 측 트레이스에 연결되지 않습니다. 클라이언트측 트레이스를 서버측 트레이스와 연결하려면 이러한 HTTP 요청에 X-Ray trace 헤더를 추가하도록 CloudWatch RUM 웹 클라이언트를 구성합니다.

### Warning

HTTP 요청에 X-Ray trace 헤더를 추가하도록 CloudWatch RUM 웹 클라이언트를 구성하면 원본 간 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 활성화](#)를 참조하십시오. 프로덕션 환경에서 클라이언트 측 X-Ray 추적 헤더를 추가하기 전에 애플리케이션을 테스트하는 것을 강력하게 권장합니다. 자세한 내용은 [CloudWatch RUM 웹 클라이언트 설명서](#)를 참조하십시오.

에서의 CloudWatch 실제 사용자 모니터링에 대한 자세한 내용은 [CloudWatch RUM 사용](#)을 참조하십시오. X-Ray로 사용자 세션을 추적하는 것을 포함하여 CloudWatch RUM을 사용하도록 애플리케이션을 [설정하려면 CloudWatch RUM을 사용하도록 애플리케이션 설정](#)을 참조하십시오.

## X-Ray를 사용한 CloudWatch 합성 카나리아 디버깅

CloudWatch Synthetics는 하루 24시간, 1분에 한 번 실행되는 스크립팅된 카나리아를 사용하여 엔드 포인트와 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
- tep-by-step S 태스크 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 관리](#)를 참조하십시오.



다음 예제는 Synthetics 카나리아가 제기하는 문제를 디버깅하기 위한 일반적인 사용 사례를 보여줍니다. 각 예제는 추적 맵 또는 X-Ray Analytics 콘솔을 사용하여 디버깅하기 위한 주요 전략을 보여줍니다.

추적 맵을 읽고 상호 작용하는 방법에 대한 자세한 내용은 [서비스 맵 보기](#)를 참조하십시오.

X-Ray Analytics 콘솔을 읽는 방법과 이 콘솔과 상호 작용하는 방법에 대한 자세한 내용은 [AWS X-Ray Analytics 콘솔과 상호 작용](#) 단원을 참조하십시오.

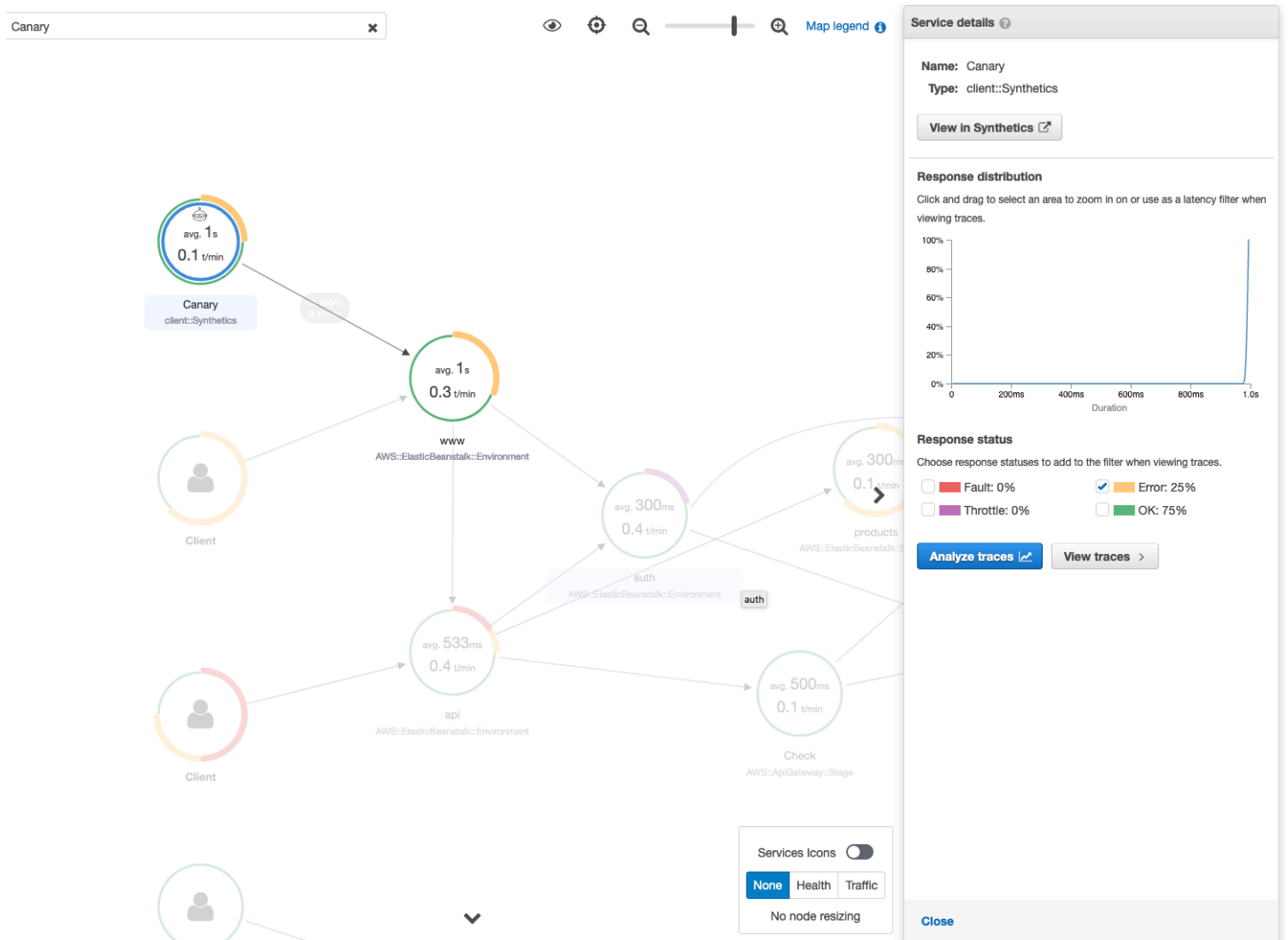
## 주제

- [추적 맵에서 오류 보고가 증가한 카나리아를 확인하세요.](#)
- [개별 트레이스의 트레이스 세부 정보 맵을 사용하여 각 요청을 자세히 볼 수 있습니다.](#)
- [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](#)

추적 맵에서 오류 보고가 증가한 카나리아를 확인하세요.

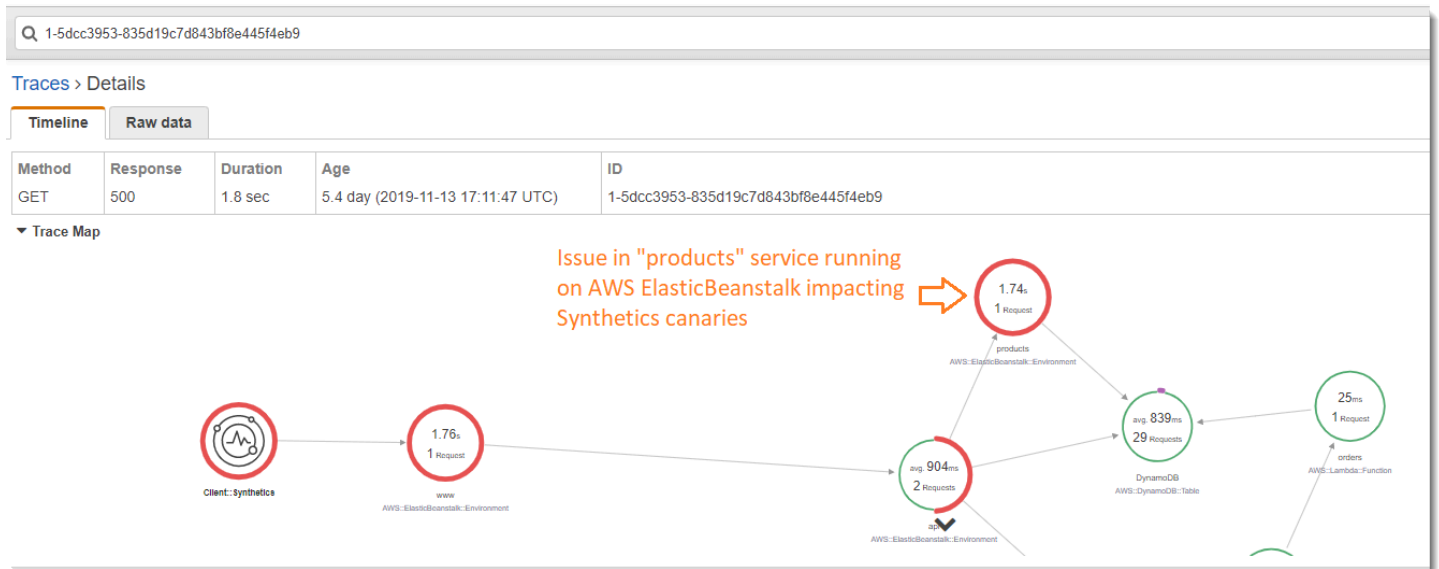
X-Ray Trace Map에서 오류, 장애, 속도 조절 속도 또는 응답 시간이 느린 카나리아를 확인하려면 필터를 사용하여 Synthetics canary 클라이언트 노드를 강조 표시할 수 있습니다. `Client::Synthetic` 자세한 정보는 [필터 표현식 사용](#)을 참조하세요. 노드를 선택하면 전체 요청의 응답 시간 분포가 표시됩니다. 두 노드 사이의 에지를 선택하면 해당 연결을 통과한 요청에 대한 세부 정보가 표시됩니다. 추적 맵에서 관련 다운스트림 서비스에 대한 “원격” 추론 노드를 볼 수도 있습니다.

Synthetics 노드를 선택하면 사이드 패널에 Synthetics에서 보기 버튼이 있어 카나리아 세부 정보를 확인할 수 있는 Synthetics 콘솔로 리디렉션됩니다.



개별 트레이스의 트레이스 세부 정보 맵을 사용하여 각 요청을 자세히 볼 수 있습니다.

지연 시간이 가장 길거나 오류를 일으키는 서비스를 확인하려면 추적 맵에서 추적을 선택하여 추적 세부 정보 맵을 호출하십시오. 개별 추적 세부 정보 맵에는 단일 요청의 end-to-end 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 canary에서 장애에 대한 CloudWatch 경보를 받으면 X-Ray의 트레이스 데이터에 대한 통계 모델링을 사용하여 X-Ray Analytics 콘솔에서 문제의 가능한 근본 원인을 파악하십시오. 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

Client

Select the node

avg. 1.2s  
1 t/min  
Canary  
client::Synthetics

Fault 67%  
1 t/min

avg. 900ms  
2 t/min  
www  
AWS::ElasticBeanstalk::Environment

avg. 533ms  
4 t/min  
api  
AWS::ElasticBeanstalk::Environment

avg. 300ms  
4 t/min  
auth  
AWS::ElasticBeanstalk::Environment

Client

Services Icons

None Health Traffic

No node resizing

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

Close

Service map

Traces

Analytics

Configuration

Sampling

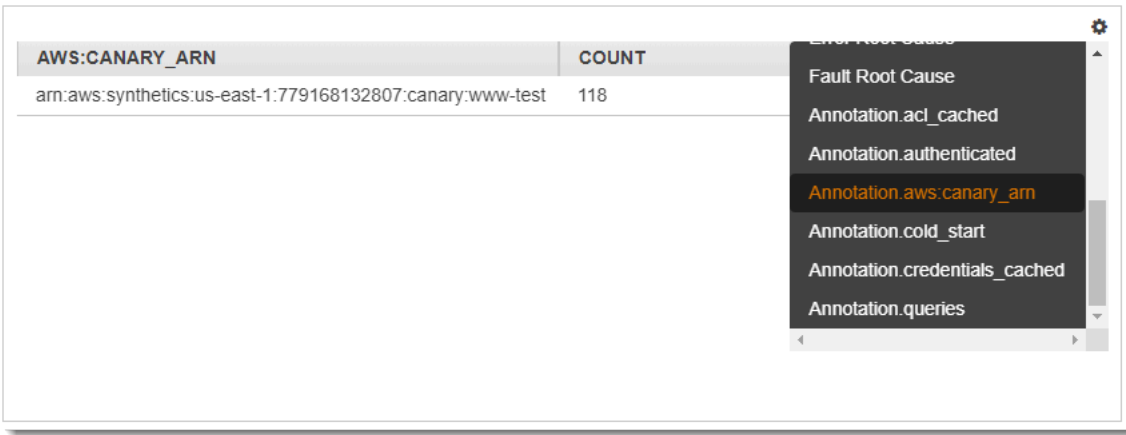
Encryption

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%

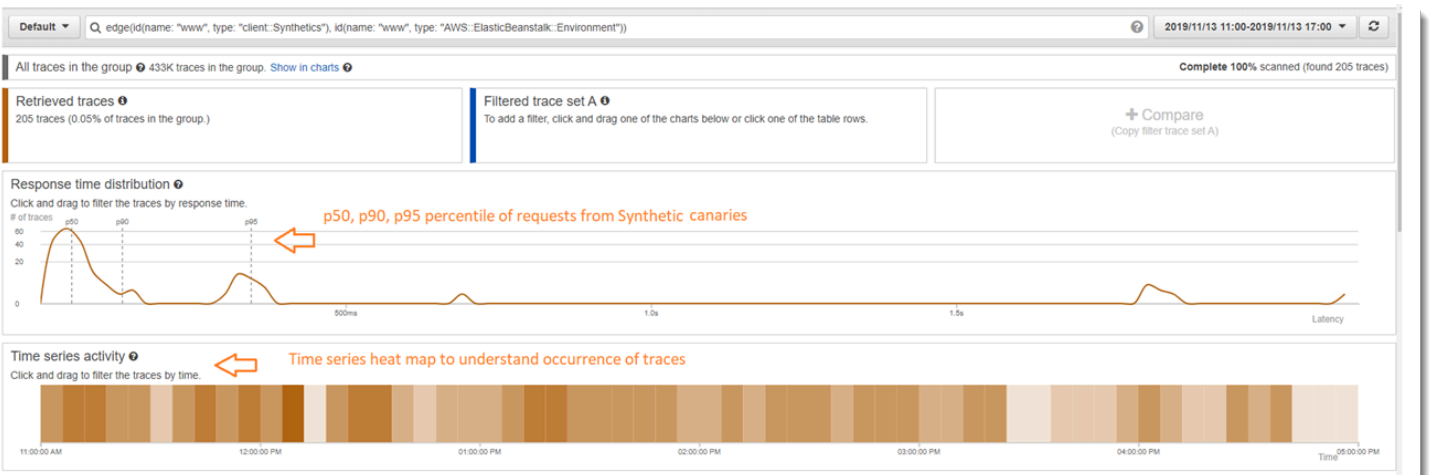
FAULT ROOT CAUSE MESSAGE	COUNT	%
ProvisionedThroughputExceededException: The level of configured provisioned throughput for the table was exceeded. Consider increasing your provisioning level with the UpdateTable API. status code: 4	4	100.00%

Root cause analysis indicating throughput capacity exceeded for DynamoDB table



### 성능 병목 현상 및 추세 파악

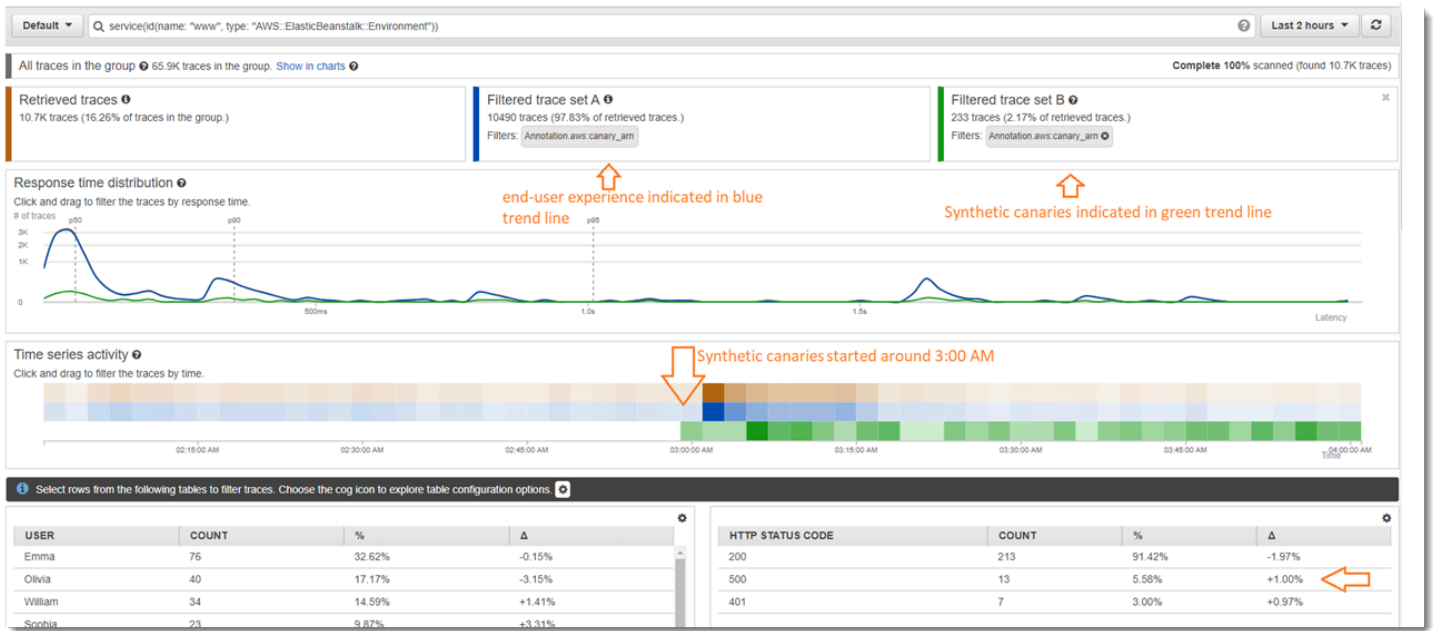
Synthetics canaries의 지속적인 트래픽을 통해 일정 기간 동안의 추적 세부 정보 맵을 채워 시간 경과에 따른 엔드포인트 성능 추세를 확인할 수 있습니다.



### 변경 전후의 지연 시간과 오류율 또는 장애율 비교

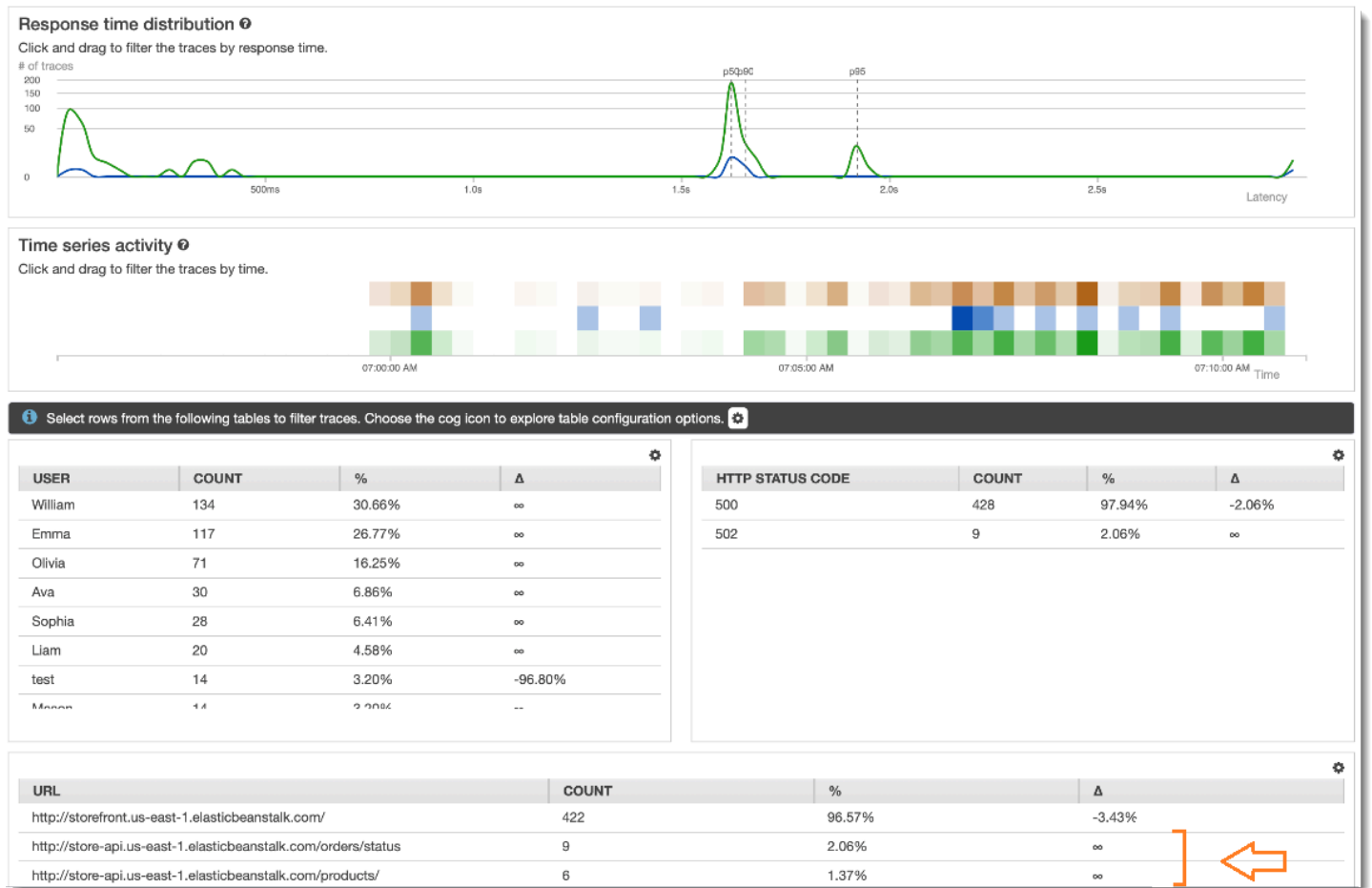
변경이 발생한 시간을 Pinpoint로 지정하여 해당 변경 사항과 카나리아에서 발견된 문제의 증가를 연관 시키십시오. X-Ray Analytics 콘솔을 사용하여 이전 및 이후 시간 범위를 다른 트레이스 세트에 정의하여 응답 시간 분포에서 시각적으로 구분할 수 있도록 만듭니다.





### 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.

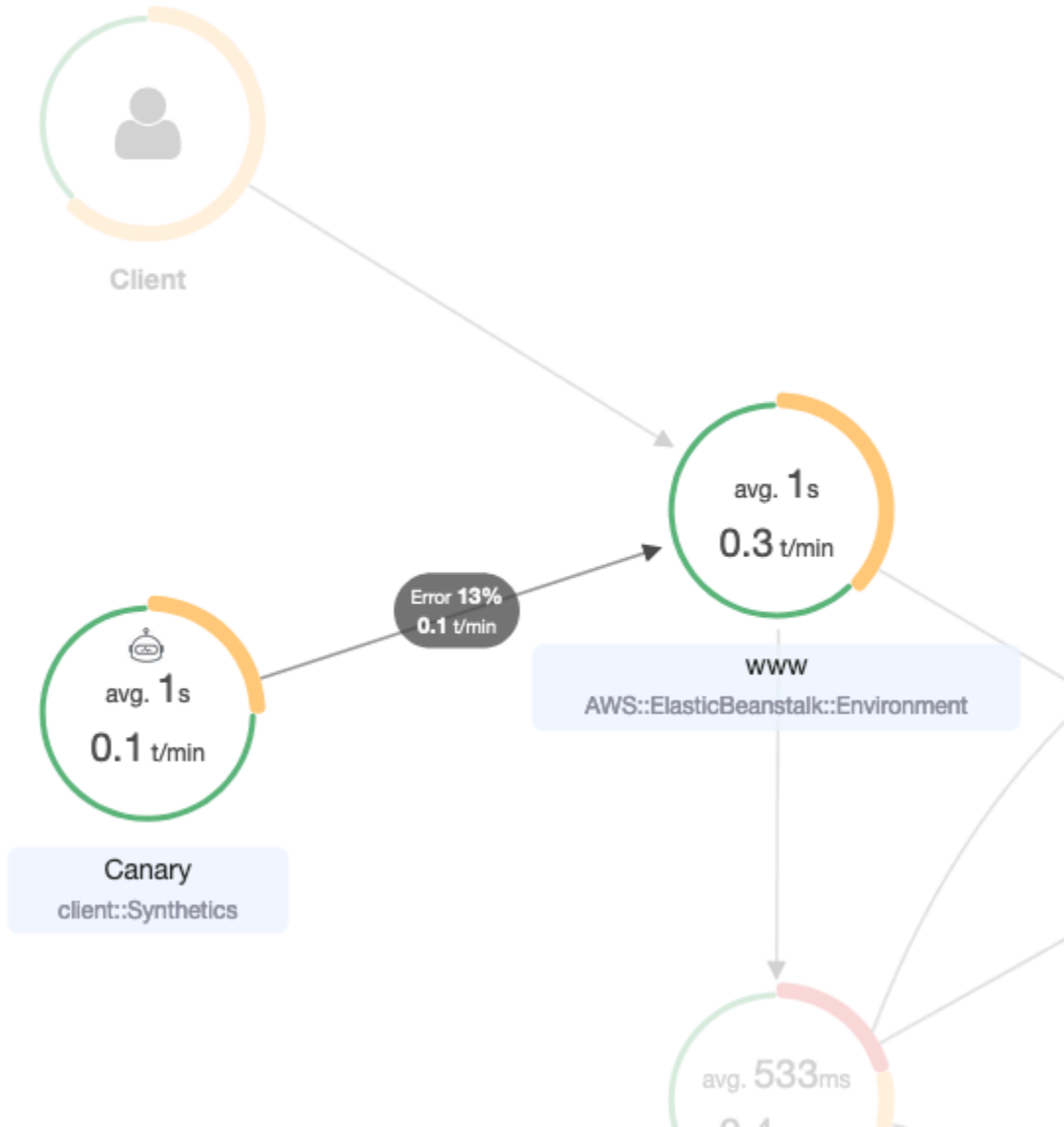


## Synthetics 테스트에 중점을 둔 그룹 사용

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()` 및 `edge()`, 를 사용하여 서비스와 엣지를 필터링할 수 있습니다. 자세한 내용은 [의 복합 키워드 섹션을 참조하십시오](#) [필터 표현식 사용](#).

### Example 그룹 필터 표현식

```
"edge(id(name: "www", type: "client::Synthetics"), id(name: "www", type: "AWS::ElasticBeanstalk::Environment"))"
```



## AWS Config를 사용하여 X-Ray 암호화 구성 변경 추적하기

AWS X-Ray는 AWS Config와 통합되어 X-Ray 암호화 리소스에 대한 구성 변경을 기록합니다. AWS Config을 사용하여 X-Ray 암호화 리소스를 인벤토리하고, X-Ray 구성 기록을 감사하고, 리소스 변경에 대한 알림을 보낼 수 있습니다.

AWS Config는 다음 X-Ray 암호화 리소스 변경을 이벤트로 로깅하는 것을 지원합니다.

- 구성 변경 - 암호화 키 변경 또는 추가, 또는 기본 X-Ray 암호화 설정으로 되돌리기.

다음 지침을 사용하여 X-Ray와 AWS Config 간 기본 연결을 생성하는 방법을 알아보십시오.

## Lambda 함수 트리거 생성

사용자 지정 AWS Config 규칙을 생성하려면 사용자 지정 AWS Lambda 함수의 ARN이 필요합니다. 다음 지침을 따라 Node.js를 사용하여 XrayEncryptionConfig 리소스의 상태를 기준으로 준수 또는 미준수 값을 AWS Config로 반환하는 기본 함수를 생성합니다.

AWS::XrayEncryptionConfig 변경 트리거를 사용하여 Lambda 함수를 생성하려면

1. [Lambda 콘솔](#)을 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 함수를 생성한 후 AWS Lambda 콘솔에 표시합니다.
5. 함수 코드를 편집하여 AWS::EC2::Instance를 AWS::XrayEncryptionConfig로 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 액세스 권한을 제공하지 않으면 규칙과 연결된 Lambda 함수를 평가할 때 AWS Config에서 범위를 벗어났다는 메시지가 표시됩니다.

```
{
  "Sid": "Stmt1529350291539",
  "Action": [
    "xray:GetEncryptionConfig"
  ],
  "Effect": "Allow",
  "Resource": "*"
}
```

## X-Ray용 사용자 지정 AWS Config 규칙 생성

Lambda 함수가 생성되면 함수의 ARN을 참조하고 AWS Config 콘솔로 이동하여 사용자 지정 규칙을 생성합니다.

X-Ray용 AWS Config 규칙을 생성하려면

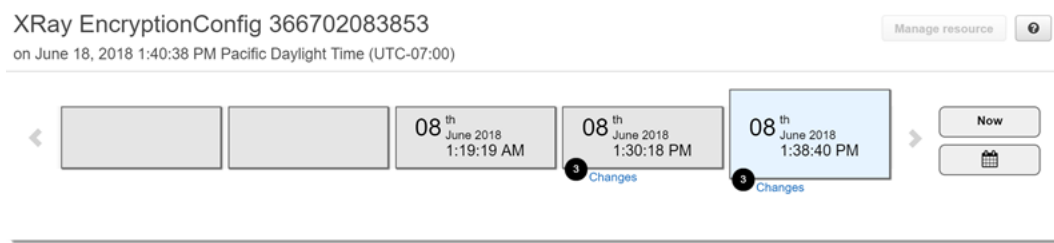
1. [AWS Config 콘솔의 규칙 페이지](#)를 엽니다.
2. 규칙 추가를 선택한 다음 사용자 지정 규칙 추가를 선택합니다.
3. AWS Lambda 함수 ARN에서 사용하려는 Lambda 함수와 연결된 ARN을 삽입합니다.
4. 설정할 트리거의 유형을 선택합니다.
  - 구성 변경 – 규칙의 범위와 일치하는 리소스의 구성이 변경되면 AWS Config가 평가를 트리거합니다. AWS Config가 구성 항목 변경 알림을 보내면 평가가 실행됩니다.
  - 주기적 – AWS Config는 사용자가 선택한 빈도(예: 매 24시간)로 이 규칙에 대한 평가를 실행합니다.
5. 리소스 유형으로는 X-Ray 섹션에서 EncryptionConfig를 선택합니다.
6. Save를 선택합니다.

AWS Config 콘솔이 규칙의 규정 준수를 즉시 평가합니다. 평가를 완료하는 데 몇 분 정도 걸릴 수 있습니다.

이제 이 규칙이 규정을 준수하므로 AWS Config가 감사 기록 컴파일을 시작할 수 있습니다. AWS Config는 타임라인 형식으로 리소스 변경을 기록합니다. 이벤트의 타임라인에 변경이 발생할 때마다 AWS Config는 전/후 형식 테이블을 생성하여 암호화 키의 JSON 표현에서 변경된 내용을 보여줍니다. EncryptionConfig와 연결된 2개의 필드 변경은 Configuration.type 및 Configuration.keyID입니다.

## 결과 예

다음은 특정 일시에 발생한 변경을 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 AWS Config를 Amazon SNS 알림을 게시하도록 설정합니다. 자세한 내용은 [이메일로 AWS Config 리소스 변경 모니터링](#) 단원을 참조하십시오.

## Amazon Elastic Compute Cloud와 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](#) 섹션을 참조하세요.

## AWS Elastic Beanstalk 및 AWS X-Ray

AWS Elastic Beanstalk 플랫폼에는 X-Ray 대몬(daemon)이 포함되어 있습니다. Elastic Beanstalk 콘솔에서 옵션을 설정하거나 구성 파일을 사용하여 [대몬\(daemon\)을 실행](#)할 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 Java는 Maven에서 제공되므로, 애플리케이션 코드만 배포하여 인스턴스 상에서 빌드할 수 있으므로 모든 종속성을 번들링하고 업로드할 필요가 없습니다.

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 구성](#) 섹션을 참조하십시오.

## Elastic Load Balancing 및 AWS X-Ray

Elastic Load Balancing 애플리케이션 로드 밸런서가 X-Amzn-Trace-Id라는 헤더에서 수신 HTTP 요청에 트레이스 ID를 추가합니다.

```
X-Amzn-Trace-Id: Root=1-5759e988-bd862e3fe1be46a994272793
```

### X-Ray 트레이스 ID 형식

X-Ray trace\_id는 하이픈으로 구분된 3개의 숫자로 구성됩니다. 예를 들어 1-58406520-a006649127e371903a2de979입니다. 여기에는 다음이 포함됩니다.

- 버전 번호, 즉1.
- 원래 요청 시간을 Unix 시대 시간으로 나타낸 8자리 16진수.

예를 들어 2016년 12월 1일 오전 10시 (PST) 기준 에포크 타임은 초 또는 16진수로 표시됩니다.  
1480615200 58406520

- 24자리 16진수 트레이스를 위한 전 세계적으로 고유한 96비트 식별자입니다.

로드 밸런서는 X-Ray로 데이터를 전송하지 않으며 서비스 맵에 노드로 표시되지 않습니다.

자세한 내용은 Elastic Load Balancing 개발자 가이드에서 [Application Load Balancer에 대한 추적 요청](#)을 참조하세요.

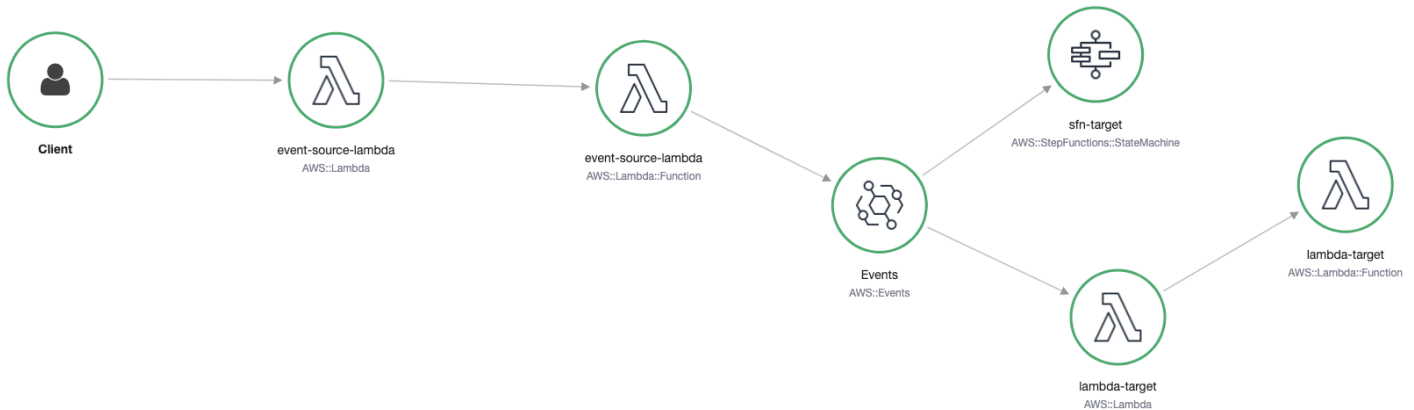
## 아마존 EventBridge 및 AWS X-Ray

AWS X-Ray Amazon과 EventBridge 통합하여 EventBridge 전달된 이벤트를 추적합니다. [X-Ray SDK로 인스트루먼트된 서비스가 이벤트를 보내는 EventBridge 경우 추적 컨텍스트는 추적 헤더 내의 다운스트림 이벤트 대상으로 전파됩니다.](#) X-Ray SDK는 자동으로 추적 헤더를 가져와 이후의 모든 계층에 적용합니다. 이러한 연속성을 통해 사용자는 다운스트림 서비스 전반을 추적, 분석, 디버깅할 수 있으며 시스템을 보다 완벽하게 파악할 수 있습니다.

자세한 내용은 EventBridge 사용자 안내서의 EventBridge [X-Ray 통합](#)을 참조하십시오.

## X-Ray 서비스 맵에서 소스 및 대상 보기

X-Ray trace 맵은 소스 및 타겟 서비스를 연결하는 EventBridge 이벤트 노드를 표시합니다. 자세한 정보는 [X-Ray 트레이스 맵 사용](#)을 참조하세요. 다음은 추적 맵의 예입니다.



## 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 함수에서의 호출을 EventBridge 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.printStackTrace();
    }
    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 Lambda 함수를 AWS X-Ray 추적하는 데 사용할 수 있습니다. Lambda는 [X-Ray 대몬\(daemon\)](#)을 실행하고 함수 호출 및 실행에 대한 세부 정보가 포함된 세그먼트를 기록합니다. 추가 구성을 위해 사용자 함수와 함께 X-Ray SDK를 번들링하여 발신 호출을 기록하고 주식 및 메타데이터를 추가할 수 있습니다.

다른 구성된 서비스에서 Lambda 함수를 호출하는 경우는 이미 샘플링된 요청을 추가 구성 없이 추적합니다. 업스트림 서비스는 구성된 웹 애플리케이션 또는 다른 Lambda 함수일 수 있습니다. 서비스는 인스트루먼트된 AWS SDK 클라이언트를 사용하여 직접 함수를 호출하거나 인스트루먼트된 HTTP 클라이언트로 API Gateway API를 호출하여 함수를 호출할 수 있습니다.

AWS X-Ray Amazon AWS Lambda SQS를 사용하는 이벤트 기반 애플리케이션 추적을 지원합니다. CloudWatch 콘솔을 사용하면 Amazon SQS로 대기하고 다운스트림 Lambda 함수로 처리되는 각 요청의 연결된 보기를 볼 수 있습니다. 업스트림 메시지 생산자의 트레이스는 다운스트림 Lambda 소비자 노드의 트레이스에 자동으로 연결되어 애플리케이션 뷰를 생성합니다. end-to-end 자세한 내용은 [이벤트 중심 애플리케이션 트레이싱](#)을 참조하세요.

### Note

다운스트림 Lambda 함수에 대해 트레이스를 활성화한 경우 다운스트림 함수를 호출하는 루트 Lambda 함수에 대해서도 트레이스를 활성화해야 다운스트림 함수가 트레이스를 생성할 수 있습니다.

Lambda 함수가 예정대로 실행되거나 구성되지 않은 서비스에서 호출되는 경우 활성 추적을 통해 호출을 샘플링하고 기록하도록 Lambda를 구성할 수 있습니다.

AWS Lambda 함수에 X-Ray 통합을 구성하려면

1. [AWS Lambda 콘솔](#)을 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. [편집(Edit)]을 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 사용](#)을 참조하십시오.

## 아마존 SNS 및 AWS X-Ray

Amazon Simple Notification Service (Amazon SNS) 와 AWS X-Ray 함께 사용하면 요청이 SNS 주제를 통해 SNS 지원 구독 서비스로 이동하는 요청을 추적하고 분석할 수 있습니다. 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 토픽에 게시하는 경우, Amazon SNS는 게시자에서 구독자에게 추적 컨텍스트를 전달합니다. 또한 액티브 트레이싱을 활성화하여 계측된 SNS 클라이언트에서 게시된 메시지에 대해 Amazon SNS 구독에 대한 세그먼트 데이터를 X-Ray로 전송할 수 있습니다.



니다. Amazon SNS 콘솔을 사용하거나 Amazon SNS API 또는 CLI를 사용하여 Amazon SNS 주제에 대한 [액티브 트레이스를 활성화합니다](#). SNS 클라이언트 계측에 대한 자세한 내용은 [애플리케이션 계측](#)을 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 예시](#)를 참조하세요. Amazon SNS 콘솔을 사용하여 액티브 추적을 켜는 방법에 대한 자세한 내용은 Amazon 단순 알림 서비스 개발자 가이드의 [Amazon SNS에서 액티브 추적 사용 설정 항목](#)을 참조하세요.

AWS CLI 또는 SDK를 사용하여 활성 추적을 활성화하는 경우 리소스 기반 정책을 사용하여 권한을 수동으로 구성해야 합니다. [PutResourcePolicy](#)를 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.

```

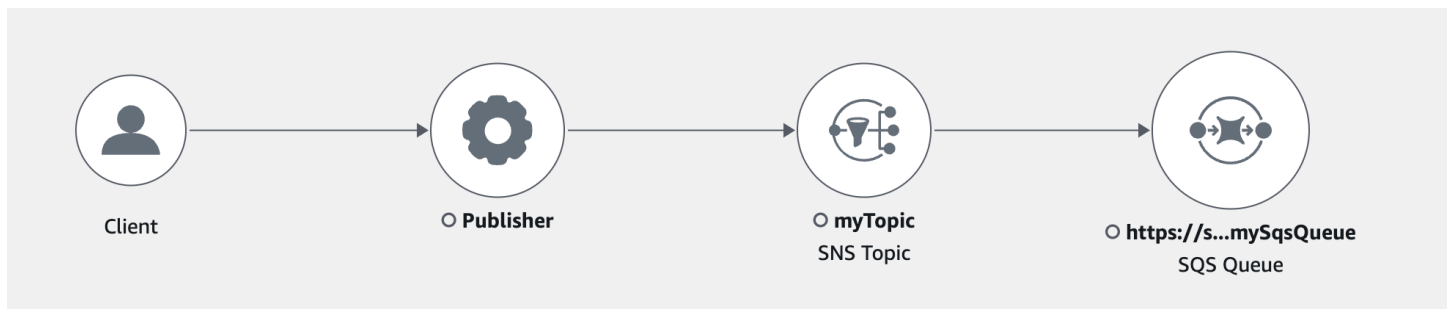
aws xray put-resource-policy --policy-name MyResourcePolicy --policy-document
'{ "Version": "2012-10-17", "Statement": [ { "Sid": "SNSAccess", "Effect": "Allow",
"Principal": { "Service": "sns.amazonaws.com" }, "Action": [ "xray:PutTraceSegments",
"xray:GetSamplingRules", "xray:GetSamplingTargets" ],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

```

이 예제를 사용하려면, *partitionregionaccount-id*, 를 특정 AWS 파티션, 지역, 계정 ID, Amazon SNS 주제 이름으로 바꾸십시오. *topic-name* 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

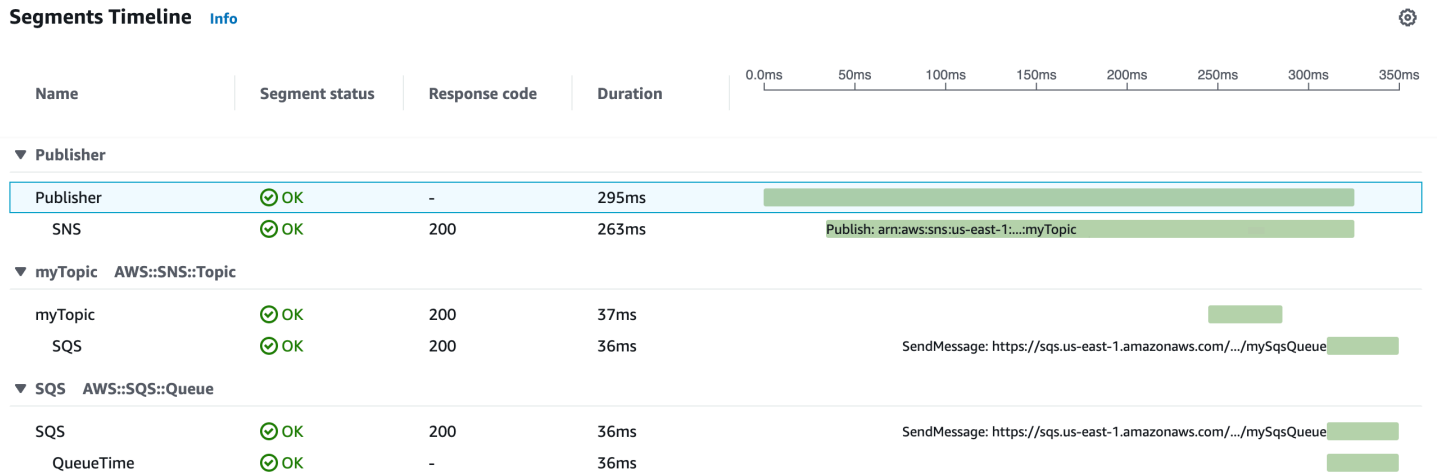
X-Ray 콘솔을 사용하여 Amazon SNS 게시자 및 구독자의 연결된 보기를 표시하는 추적 맵과 추적 세부 정보를 볼 수 있습니다. 주제에 대해 Amazon SNS 활성 추적이 켜지면 X-Ray 추적 맵 및 추적 세부 정보 맵에 Amazon SNS 게시자의 연결 노드, Amazon SNS 주제 및 다운스트림 구독자가 표시됩니다.



Amazon SNS 게시자와 구독자를 아우르는 트레이스를 선택하면 X-Ray trace details 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.



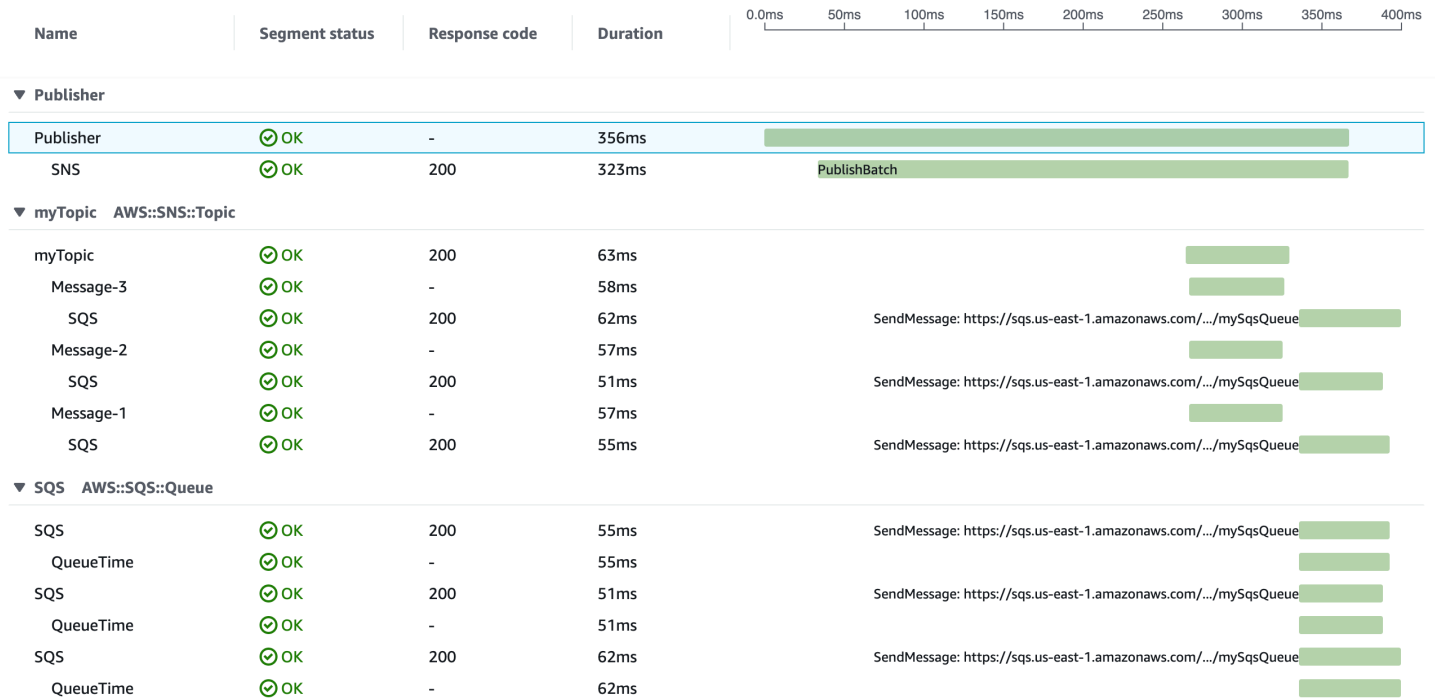
위의 예제 타임라인은 Amazon SNS 메시지 흐름에 대한 세부 정보를 제공합니다:

- 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.

Segments Timeline [Info](#)



## AWS Step Functions 및 AWS X-Ray

AWS X-Ray은 AWS Step Functions와 통합되어 Step Function에 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 Functions](#)를 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 Tracing을 활성화할 수 없습니다. 대신 상태 시스템을 만든 후 X-Ray Tracing을 활성화하세요.
4. 다음(Next)을 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. 편집(Edit)을 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.

**Permissions**

**Execution role**  
The IAM role that defines which resources your state machine has permission to access during execution. To create a custom role, go to the [IAM console](#).

Create new role  
Let Step Functions create a new role for you based on your state machine's definition and configuration details.

Choose an existing role

Enter a role ARN

5. Save를 선택합니다.

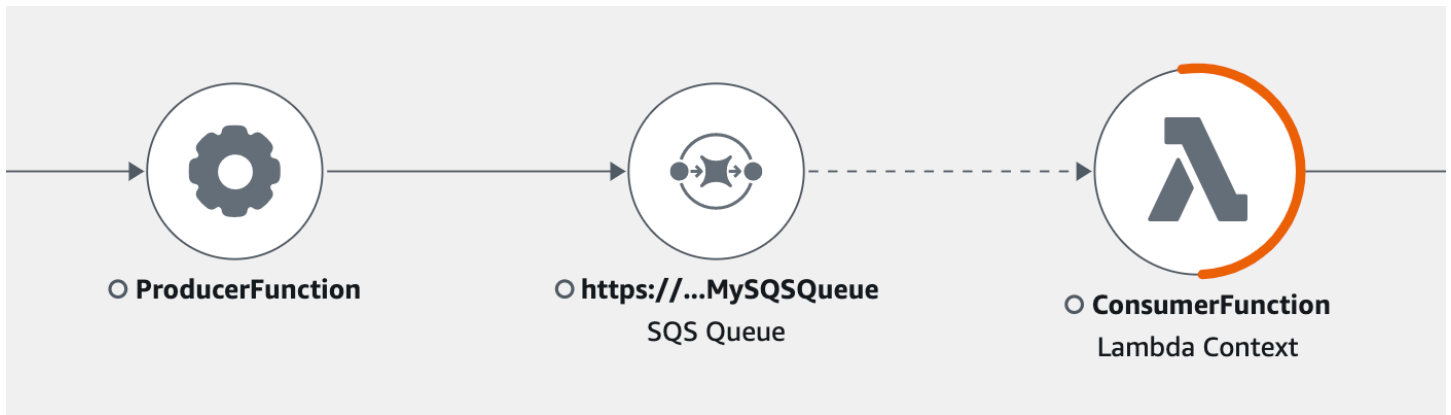
### i Note

새 상태 시스템을 생성할 때 요청이 샘플링되고 Amazon API Gateway 또는 AWS Lambda과 같은 업스트림 서비스에서 추적이 활성화되면 자동으로 추적됩니다. 콘솔을 통해 구성되지 않은 기존 상태 머신(예: AWS CloudFormation 템플릿)의 경우, X-Ray 추적을 활성화할 수 있는 충분한 권한을 부여하는 IAM 정책이 있는지 확인하세요.

## 아마존 SQS 및 AWS X-Ray

AWS X-Ray Amazon 심플 큐 서비스 (Amazon SQS) 와 통합되어 Amazon SQS 대기열을 통해 전달되는 메시지를 추적합니다. X-Ray SDK를 사용하여 요청을 추적하는 서비스가 있는 경우, Amazon SQS 는 추적 헤더를 전송하고 발신자가 소비자에게 보낸 원본 추적에 일정한 추적 ID를 계속해서 전파합니다. 사용자는 이러한 추적의 연속성 덕분에 다운스트림 서비스 전체에 걸쳐 추적, 분석 및 디버깅할 수 있습니다.

AWS X-Ray Amazon SQS 및 를 사용하는 이벤트 기반 애플리케이션 추적을 지원합니다. AWS Lambda CloudWatch 콘솔을 사용하면 Amazon SQS로 대기하고 다운스트림 Lambda 함수로 처리되는 각 요청의 연결된 보기를 볼 수 있습니다. 업스트림 메시지 생산자의 트레이스는 다운스트림 Lambda 소비자 노드의 트레이스에 자동으로 연결되어 애플리케이션 뷰를 생성합니다. end-to-end 자세한 내용은 [이벤트 중심 애플리케이션 트레이싱](#)을 참조하세요.



Amazon SQS는 다음과 같은 추적 헤더 계층을 지원합니다.

- 기본 HTTP 헤더 — SDK를 통해 Amazon SQS를 호출하면 X-Ray SDK가 자동으로 추적 헤더를 HTTP 헤더로 채웁니다. AWS 기본 추적 헤더는 X-Amzn-Trace-Id로 입력되며, [SendMessage](#) 또는 [SendMessageBatch](#) 요청에 포함된 모든 메시지에 해당합니다. 기본 HTTP 헤더에 대한 자세한 내용은 [추적 헤더](#) 단원을 참조하십시오.
- **AWSTraceHeader** 시스템 속성 — AWSTraceHeader는 대기열의 메시지와 함께 X-Ray 추적 헤더를 전달하기 위해 Amazon SQS에서 사용하는 [메시지 시스템 속성](#)입니다. 예를 들어 새로운 언어에 대한 추적 SDK를 빌드할 때와 같이 X-Ray SDK를 통한 자동 계층이 제공되지 않는 경우에도 AWSTraceHeader를 사용할 수 있습니다. 두 가지 헤더 계층을 설정하고 나면 그 메시지 시스템 속성이 HTTP 추적 헤더를 덮어씁니다.

Amazon EC2에서 실행하는 경우 Amazon SQS는 한 번에 하나의 메시지 처리를 지원합니다. 이는 온 프레미스 호스트에서 실행할 때와 Amazon ECS 또는 같은 AWS Fargate 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. 할당량에 대해 자세히 알아보려면 [Amazon AWS SQS 할당량을 참조하십시오](#).

## HTTP 추적 헤더 전송

Amazon SQS의 발신자 구성 요소는 [SendMessageBatch](#) 또는 [SendMessage](#) 직접 호출을 통해 추적 헤더를 자동 전송할 수 있습니다. AWS SDK 클라이언트를 계층하면 X-Ray SDK를 통해 지원되는 모든 언어를 통해 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 SDK의 다음 주제를 참조하십시오.

- Go – [Go용 X-Ray AWS SDK를 사용하여 SDK 호출 추적하기](#)
- Java – [Java용 X-Ray AWS SDK를 사용하여 SDK 호출 추적하기](#)
- Node.js – [Node.js 용 X-Ray AWS SDK를 사용하여 SDK 호출 추적하기](#)
- Python – [Python용 X-Ray AWS SDK를 사용하여 SDK 호출 추적하기](#)
- Ruby – [Ruby용 X-Ray AWS SDK를 사용하여 SDK 호출 추적하기](#)
- .NET – [.NET용 X-Ray AWS SDK를 사용하여 SDK 호출을 추적합니다.](#)

## 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 계층해야 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));
}
}
```

## 아마존 S3와 AWS X-Ray

AWS X-Ray Amazon S3와 통합하여 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 추적 맵을 사용하여 Amazon S3와 애플리케이션이 사용하는 다른 서비스 간의 연결을 볼 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 살펴보기](#) 섹션을 참조하십시오.

Amazon S3는 기본 http 헤더 계층을 지원합니다. SDK를 통해 Amazon S3를 호출하면 X-Ray SDK가 자동으로 추적 헤더를 HTTP 헤더로 채웁니다. AWS 기본 추적 헤더는 X-Amzn-Trace-Id로 전달됩니다. 추적 헤더에 대한 자세한 내용은 개념 페이지의 [추적 헤더](#)를 참조하십시오. Amazon S3 추적 컨텍스트 전파는 다음 구독자를 지원합니다: Lambda, SQS 및 SNS. SQS와 SNS는 세그먼트 데이터를 자체적으로 내보내지 않기 때문에 S3에 의해 트리거될 때 추적 헤더를 다운스트림 서비스에 전파하더라도 추적 또는 추적 맵에 표시되지 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 유형](#)을 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 AWS Identity and Access Management 대기열에 (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 [아마존 SNS 및 AWS X-Ray](#) 및 [아마존 SQS 및 AWS X-Ray](#)을 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.

Amazon S3에 Lambda 함수를 AWS Lambda 호출할 권한을 부여할 수도 있습니다. 자세한 내용은 [AWS Lambda 개발자 안내서의 자습서: Amazon S3와 Lambda 사용](#)을 참조하십시오. AWS

Lambda와 X-Ray를 통합하는 방법에 대한 자세한 내용은 Lambda에서의 자바 [코드 계측을 참조하십시오](#). AWS

## X-Ray에서 리소스 관리

이 가이드에서는 키 및 선택적 값 쌍을 사용하여 리소스를 구성하고 리소스에 태그를 지정하여 템플릿을 사용하여 X-Ray에서 리소스를 관리하는 방법을 보여줍니다.

[AWS CloudFormation 템플릿](#)을 사용하여 리소스와 인프라를 자동으로 설정하고 관리할 수 있습니다. 이 템플릿을 두 JSON 가지 YAML 형식 중 하나로 사용하여 단일 파일에 X-Ray 그룹, 샘플링 규칙 또는 리소스 정책을 만들 수 있습니다. 예를 들어 다음과 같은 용도로 AWS CloudFormation 템플릿을 사용할 수 있습니다.

- 단일 템플릿을 사용하여 여러 배포에서 리소스를 일관되게 구성하고 수동 구성 오류를 방지할 수 있습니다.
- 템플릿 파일을 사용하여 AWS 계정, 개발 환경 전반의 리소스를 관리하고 팀 간에 템플릿 파일을 공유할 수 있습니다.
- 버전 관리를 사용하여 템플릿의 변경 내용을 제어 및 추적하고 필요한 경우 변경 내용을 되돌릴 수 있습니다.

또한 리소스에 태그를 할당하여 태그를 기반으로 리소스를 검색 및 필터링하고 태그 기반 권한을 적용할 수 있도록 할 수 있습니다. 예를 들어 다음과 같은 경우에 태그를 사용할 수 있습니다.

- 특정 팀, 부서 또는 애플리케이션 용도에 태그를 지정하여 이들이 사용하는 리소스를 추적하세요.
- 민감한 문서와 같이 특별한 처리가 필요한 리소스에는 태그를 지정하세요.
- 피크 시간대에는 사용을 중지하도록 스크립트를 사용하여 태그가 지정된 리소스를 자동으로 관리합니다.

다음 섹션에서는 AWS CloudFormation 템플릿과 태그가 지정된 리소스를 사용하여 리소스를 관리하는 방법에 대한 추가 정보를 제공합니다.

### 주제

- [AWS CloudFormation로 X-Ray 리소스 만들기](#)
- [X-Ray 샘플링 규칙 및 그룹 태그 지정하기](#)

## AWS CloudFormation로 X-Ray 리소스 만들기

AWS X-Ray 와 [AWS CloudFormation](#) 통합되어 리소스를 모델링하고 설정하는 데 도움이 되므로 AWS 리소스와 인프라를 만들고 관리하는 데 소요되는 시간을 줄일 수 있습니다. 사용하려는 모든 AWS 리소스를 설명하는 템플릿을 만들고 해당 리소스를 자동으로 AWS CloudFormation 프로비저닝 및 구성합니다.

를 사용하면 AWS CloudFormation 템플릿을 재사용하여 일관되고 반복적으로 X-Ray 리소스를 설정할 수 있습니다. 리소스를 한 번 설명한 다음 여러 AWS 계정 지역과 지역에 동일한 리소스를 반복해서 프로비저닝하세요.

### X-Ray 및 AWS CloudFormation 템플릿

X-Ray 및 관련 서비스를 위한 리소스를 프로비저닝하고 구성하려면 [AWS CloudFormation 템플릿](#) 을 사용하십시오. 템플릿은 JSON 또는 YAML 형식의 텍스트 파일입니다. 이러한 템플릿은 스택에 프로비저닝하려는 리소스를 설명합니다. AWS CloudFormation JSON이나 YAML에 익숙하지 않은 경우 AWS CloudFormation Designer를 사용하여 템플릿을 시작하는 데 도움을 받을 수 있습니다. AWS CloudFormation 자세한 내용은 AWS CloudFormation 사용 설명서에서 [AWS CloudFormation Designer이란 무엇입니까?](#)를 참조하세요.

X-Ray는 [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 참조](#)를 참조하세요.

### 에 대해 자세히 알아보기 AWS CloudFormation

자세히 AWS CloudFormation 알아보려면 다음 리소스를 참조하십시오.

- [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 문구입니다. 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 리소

스를 검색하고 필터링할 수 있습니다. 태그에 대한 자세한 내용은 AWS 일반 참조 안내서의 [AWS 리소스 태깅](#)을 참조하세요.

태그를 사용하여 배포에 태그 기반 권한을 적용할 수 있습니다 CloudFront . 자세한 내용은 [리소스 태그를 사용한 AWS 리소스 액세스 제어](#)를 참조하세요.

#### Note

[Tag Editor](#) 및 [AWS Resource Groups](#)는 현재 X-Ray 리소스를 지원하지 않습니다. AWS X-Ray 콘솔이나 API를 사용하여 태그를 추가하고 관리할 수 있습니다.

X-Ray 콘솔, API AWS CLI, SDK 등을 사용하여 리소스에 태그를 적용할 수 있습니다. 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 — 명령 참조의 [xray](#)를 AWS CLI 참조하십시오.
- 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 권한을 가진 기업 내 AWS 사용자에게 문의하십시오.

## 주제

- [태그 제한](#)
- [콘솔에서 태그 관리](#)
- [내 태그 관리 AWS CLI](#)

- [태그를 기반으로 X-Ray 리소스에 대한 액세스 제어](#)

## 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

- [새 그룹에 태그 추가 \(콘솔\)](#)
- [새 샘플링 규칙에 태그 추가하기 \(콘솔\)](#)
- [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. <https://console.aws.amazon.com/xray/home> 에서 AWS Management Console 로그인하고 X-Ray 콘솔을 엽니다.
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 제한](#)을 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. <https://console.aws.amazon.com/xray/home> 에서 AWS Management Console 로그인하고 X-Ray 콘솔을 엽니다.
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 규칙을 구성합니다](#) 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 제한](#)을 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. <https://console.aws.amazon.com/xray/home> 에서 AWS Management Console 로그인하고 X-Ray 콘솔을 엽니다.
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 구성](#)을 참조하십시오. 태그에 대한 제한 사항은 [태그 제한](#)을 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. <https://console.aws.amazon.com/xray/home> 에서 AWS Management Console 로그인하고 X-Ray 콘솔을 엽니다.
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 규칙을 구성합니다.](#)을 참조하십시오. 태그에 대한 제한 사항은 [태그 제한](#)을 참조하십시오.
5. 태그를 삭제하려면 태그 오른쪽에 있는 X를 선택합니다.
6. 태그 편집 또는 삭제를 마치면 샘플링 규칙 업데이트를 선택합니다.

## 내 태그 관리 AWS CLI

X-Ray 그룹 또는 샘플링 규칙을 만들 때 태그를 추가할 수 있습니다. 를 AWS CLI 사용하여 태그를 만들고 관리할 수도 있습니다. 기존 그룹 또는 샘플링 규칙의 태그를 업데이트하려면 AWS X-Ray 콘솔 [TagResource](#) 또는 [UntagResource](#) API를 사용합니다.

### 주제

- [새 X-Ray 그룹 또는 샘플링 규칙\(CLI\)에 태그 추가하기](#)
- [기존 리소스에 태그 추가 \(CLI\)](#)
- [리소스의 태그 나열하기 \(CLI\)](#)
- [리소스에서 태그 삭제하기 \(CLI\)](#)

## 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 [X-Ray API를 사용하여 샘플링, 그룹 및 암호화 설정 구성](#) 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 [X-Ray API를 사용하여 샘플링, 그룹 및 암호화 설정 구성](#) 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

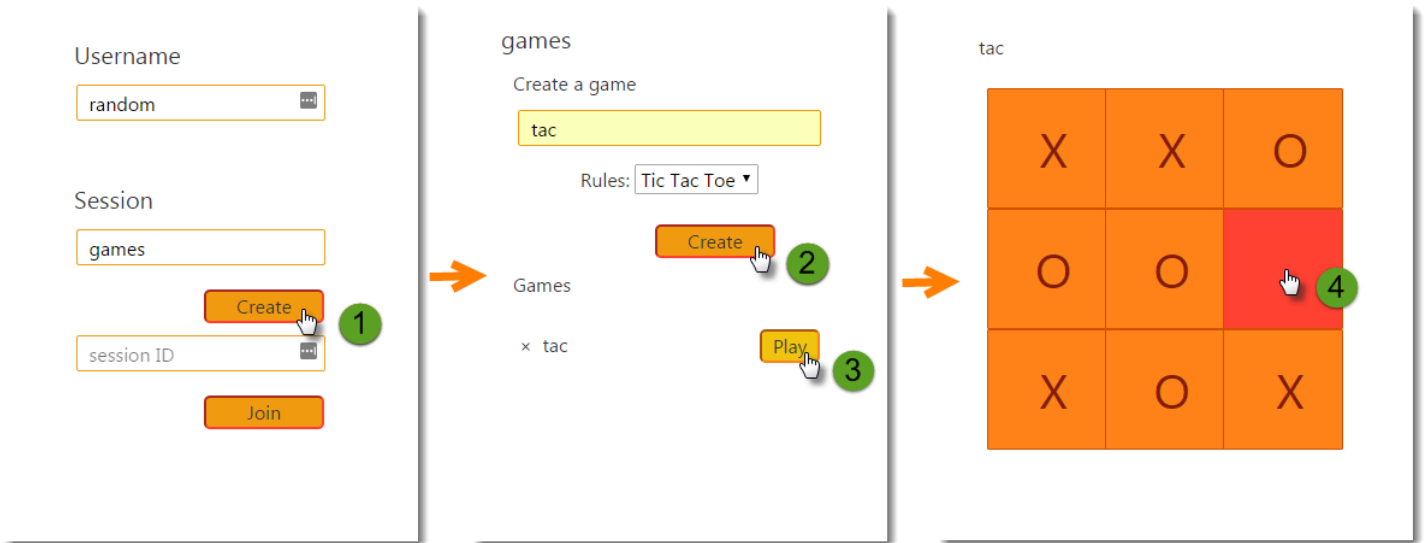
X-Ray 그룹이나 샘플링 규칙에 태그를 첨부하거나 요청에 포함된 태그를 X-Ray에 전달할 수 있습니다. 태그를 기반으로 액세스를 제어하려면 `xray:ResourceTag/key-name`, `aws:RequestTag/key-name` 또는 `aws:TagKeys` 조건 키를 사용하여 정책의 [조건 요소](#)에 태그 정보를 제공합니다. 이러한 조건 키에 대해 자세히 알아보려면 [리소스 태그를 사용한 AWS 리소스 액세스 제어를](#) 참조하십시오.

리소스의 태그를 기반으로 리소스에 대한 액세스를 제한하는 자격 증명 기반 정책의 예제는 [태그에 기반한 X-Ray 그룹 및 샘플링 규칙에 대한 액세스 관리](#) 섹션에서 확인할 수 있습니다.

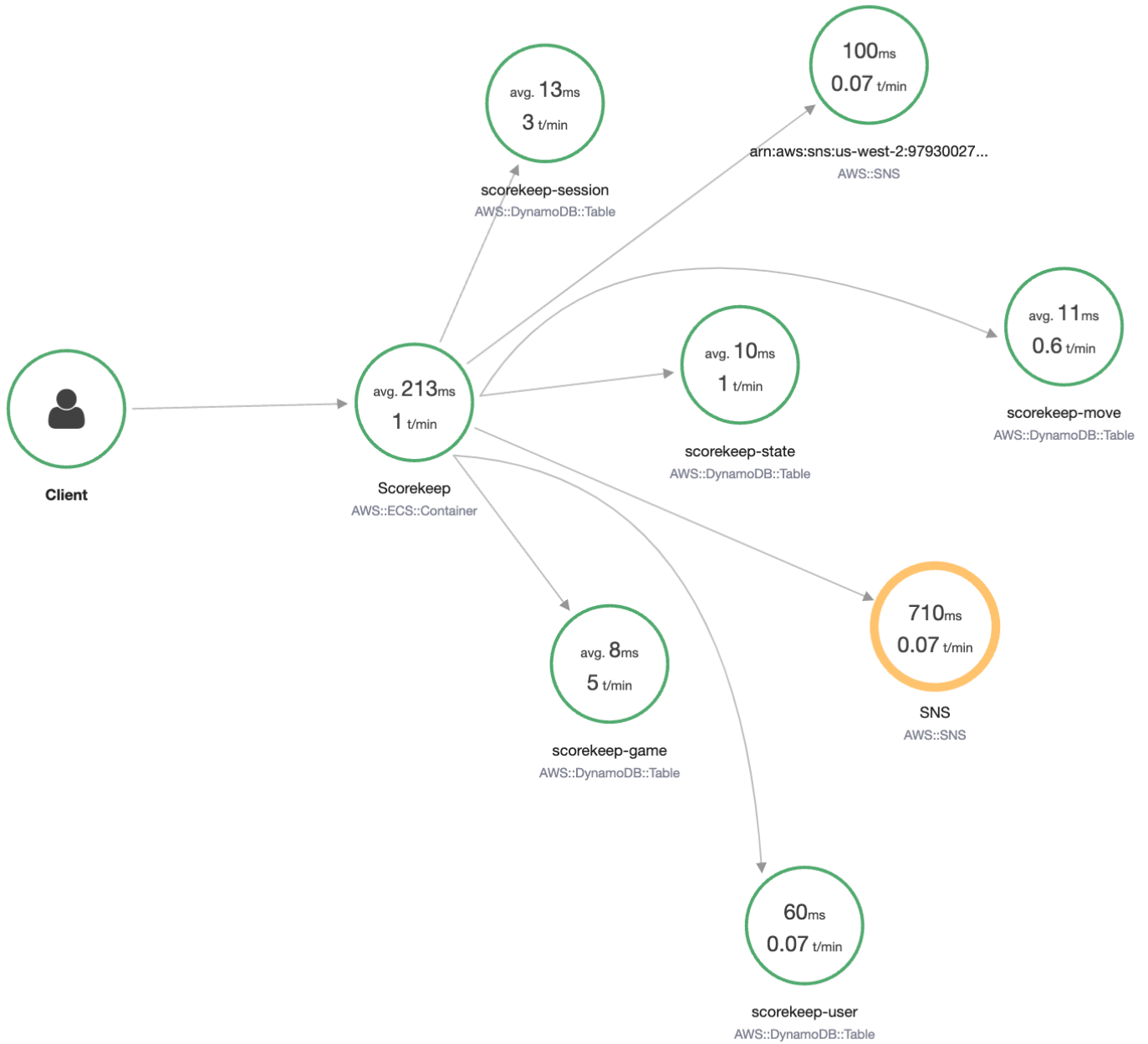
## AWS X-Ray 샘플 애플리케이션

에서 사용할 수 있는 AWS X-Ray [eb-java-scorekeep](#) 샘플 앱은 수신되는 HTTP 호출 GitHub, DynamoDB SDK 클라이언트 및 HTTP 클라이언트를 계측하기 위한 AWS X-Ray SDK를 사용하는 방법을 보여줍니다. 샘플 앱은 DynamoDB 테이블을 생성하고, 인스턴스에서 Java 코드를 컴파일하고, 추가 구성 없이 X-Ray 데몬을 실행하는 데 사용합니다 AWS CloudFormation .

또는 를 사용하여 계측된 샘플 애플리케이션을 설치하고 사용하려면 [Scorekeep 자습서](#)를 참조하십시오. AWS Management Console AWS CLI



여기에는 프론트엔드 웹 앱, 이 앱이 직접 호출하는 API, 이 앱이 데이터를 저장하는 데 사용하는 DynamoDB 테이블이 포함되어 있습니다. [필터](#), [플러그인](#), [계측된 AWS SDK 클라이언트를 사용한 기본 계측은 프로젝트 브랜치에 표시됩니다](#). `xray-gettingstarted` 이는 [시작하기 자습서](#)에서 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.

브라우저에서의 사용자 로그인 및 AWS SDK for JavaScript 사용을 지원하기 위해 `xray-cognito` 브랜치에서는 Amazon Cognito를 추가하여 사용자 인증 및 권한 부여를 지원합니다. Amazon Cognito에서 가져온 보안 인증 정보를 사용하여 웹 앱은 트레이스 데이터를 X-Ray에 보내 클라이언트 관점에서의 요청 정보를 레코딩합니다. 브라우저 클라이언트는 추적 맵에 자체 노드로 나타나며 사용자가 보고 있는 페이지의 URL, 사용자 ID 등 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. 이벤트에 CloudWatch 의해 트리거되는 Lambda 작업자는 몇 분마다 대기열에서 항목을 가져와서 처리하여 분석을 위해 Amazon S3에 게임 기록을 저장합니다.

## 주제

- [Scorekeep 샘플 애플리케이션 시작하기](#)
- [SDK 클라이언트 수동 계측 AWS](#)
- [추가 하위 세그먼트 생성](#)
- [주석, 메타데이터 및 사용자 ID 기록](#)
- [발신 HTTP 호출 구성](#)
- [PostgreSQL 데이터베이스에 대한 호출 구성](#)
- [인스트루먼트 함수 AWS Lambda](#)
- [시작 코드 구성](#)
- [스크립트 구성](#)
- [웹 앱 클라이언트 구성](#)
- [작업자 스레드에서 구성된 클라이언트 사용](#)

## Scorekeep 샘플 애플리케이션 시작하기

이 자습서에서는 Amazon ECS에서 [샘플 애플리케이션과 X-Ray AWS CloudFormation 데몬을 실행하는 리소스를 생성하고 구성하는 데 사용하는 Scorekeep](#) 샘플 애플리케이션의 `xray-gettingstarted` 브랜치를 사용합니다. 애플리케이션은 Spring 프레임워크를 사용하여 JSON 웹 API를 구현하고 Amazon DynamoDB에 데이터를 유지합니다. AWS SDK for Java 애플리케이션의 서블릿 필터는 애플리케이션이 처리하는 모든 수신 요청을 계측하고 AWS SDK 클라이언트의 요청 핸들러는 DynamoDB에 대한 다운스트림 호출을 계측합니다.

또는 `awscli`를 사용하여 이 자습서를 따를 수 있습니다. AWS Management Console AWS CLI

## Sections

- [필수 조건](#)
- [다음을 사용하여 Scorekeep 애플리케이션을 설치합니다. CloudFormation](#)
- [데이터 추적 생성](#)
- [에서 트레이스 맵을 확인하세요. AWS Management Console](#)
- [Amazon SNS 알림 구성](#)
- [샘플 애플리케이션 탐색](#)
- [선택 사항: 최소 권한 정책](#)
- [정리](#)
- [다음 단계](#)

## 필수 조건

이 자습서에서는 샘플 애플리케이션과 X-Ray 데몬을 실행하는 리소스를 만들고 구성하는 AWS CloudFormation 데 사용합니다. 튜토리얼을 설치하고 실행하려면 다음 사전 요구 사항이 필요합니다:

1. 권한이 제한된 IAM 사용자를 사용하는 경우 [IAM 콘솔](#)에서 다음 사용자 정책을 추가하십시오.
  - `AWSCloudFormationFullAccess`— 액세스 및 사용 CloudFormation
  - `AmazonS3FullAccess`— 를 CloudFormation 사용하여 템플릿 파일을 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 맵 및 트레이스를 볼 수 있는 권한
2. 를 사용하여 자습서를 실행하려면 [CLI 버전 2.7.9 이상을 설치하고](#) 이전 단계의 사용자와 함께 [CLI](#) 를 구성하십시오. AWS CLI 사용자와 함께 를 구성할 때 지역이 구성되어 있는지 확인하십시오. AWS CLI 지역이 구성되지 않은 경우 모든 CLI 명령에 `--region AWS-REGION`을 추가해야 합니다.
3. 샘플 애플리케이션 리포지토리를 복제하려면 [Git](#)이 설치되어 있어야 합니다.

- 다음 코드 예제를 사용하여 Scorekeep 리포지토리의 xray-gettingstarted 브랜치를 복제하세요.

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

다음을 사용하여 Scorekeep 애플리케이션을 설치합니다. CloudFormation

### AWS Management Console

를 사용하여 샘플 애플리케이션을 설치합니다. AWS Management Console

- [CloudFormation 콘솔](#)을 엽니다
- 스택 생성을 선택한 다음 드롭다운 메뉴에서 새 리소스 사용을 선택합니다.
- 템플릿 지정 섹션에서 템플릿 파일 업로드를 선택합니다.
- 파일 선택을 선택하고 git repo를 복제할 때 생성된 xray-scorekeep/cloudformation 폴더로 이동한 다음 cf-resources.yaml 파일을 선택합니다.
- 다음을 선택하여 계속 진행합니다.
- 스택 이름 입력란에 scorekeep을 입력하고 페이지 하단에서 다음을 선택하여 계속 진행합니다. 참고로 이 튜토리얼의 나머지 부분에서는 스택의 이름이 scorekeep으로 지정되었다고 가정합니다.
- 스택 옵션 구성 페이지의 하단으로 스크롤하여 다음을 선택하여 계속 진행합니다.
- 검토 페이지 하단으로 스크롤하여 사용자 지정 이름으로 IAM 리소스를 생성할 CloudFormation 수 있음을 확인하는 확인란을 선택하고 Create stack 을 선택합니다.
- 이제 CloudFormation 스택이 생성되고 있습니다. 스택 상태는 약 5분 동안 CREATE\_IN\_PROGRESS로 유지되다가 CREATE\_COMPLETE로 변경됩니다. 상태는 주기적으로 새로 고쳐지며, 페이지를 새로 고칠 수도 있습니다.

### AWS CLI

를 사용하여 샘플 애플리케이션을 설치합니다. AWS CLI

- 튜토리얼 앞부분에서 복제한 xray-scorekeep 리포지토리의 cloudformation 폴더로 이동합니다:



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

- 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
```

- CloudFormation 스택 상태가 CREATE\_COMPLETE 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 URL을 찾습니다. AWS Management Console

- [CloudFormation 콘솔](#)을 엽니다
- Stacks 목록에서 scorekeep을 선택합니다.
- scorekeep 스택 페이지에서 출력 탭을 선택하고 LoadBalancerUrl URL 링크를 선택하여 웹 애플리케이션을 엽니다.

### AWS CLI

를 사용하여 애플리케이션 URL을 찾으십시오. AWS CLI

- 다음 명령어를 사용하여 웹 애플리케이션의 URL을 표시합니다:

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

- 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.

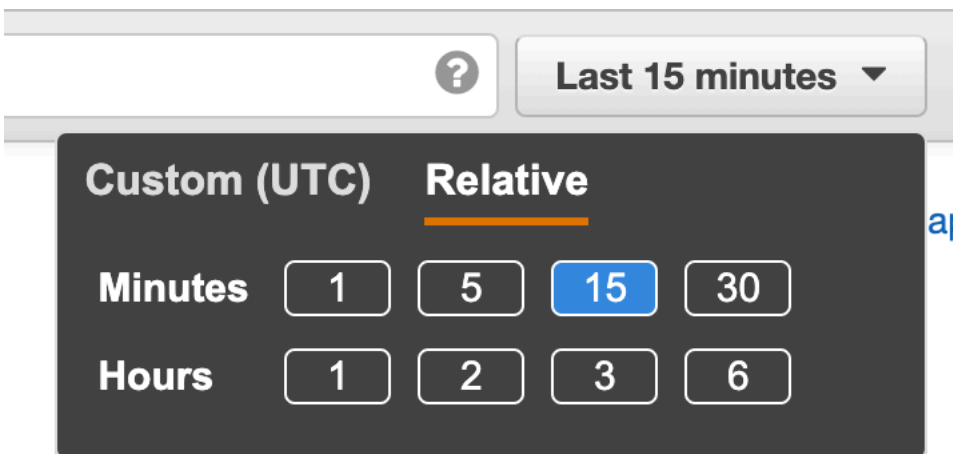
## 에서 트레이스 맵을 확인하세요. AWS Management Console

샘플 애플리케이션에서 생성된 트레이스 맵과 트레이스를 X-Ray 및 CloudWatch 콘솔에서 볼 수 있습니다.

### X-Ray console

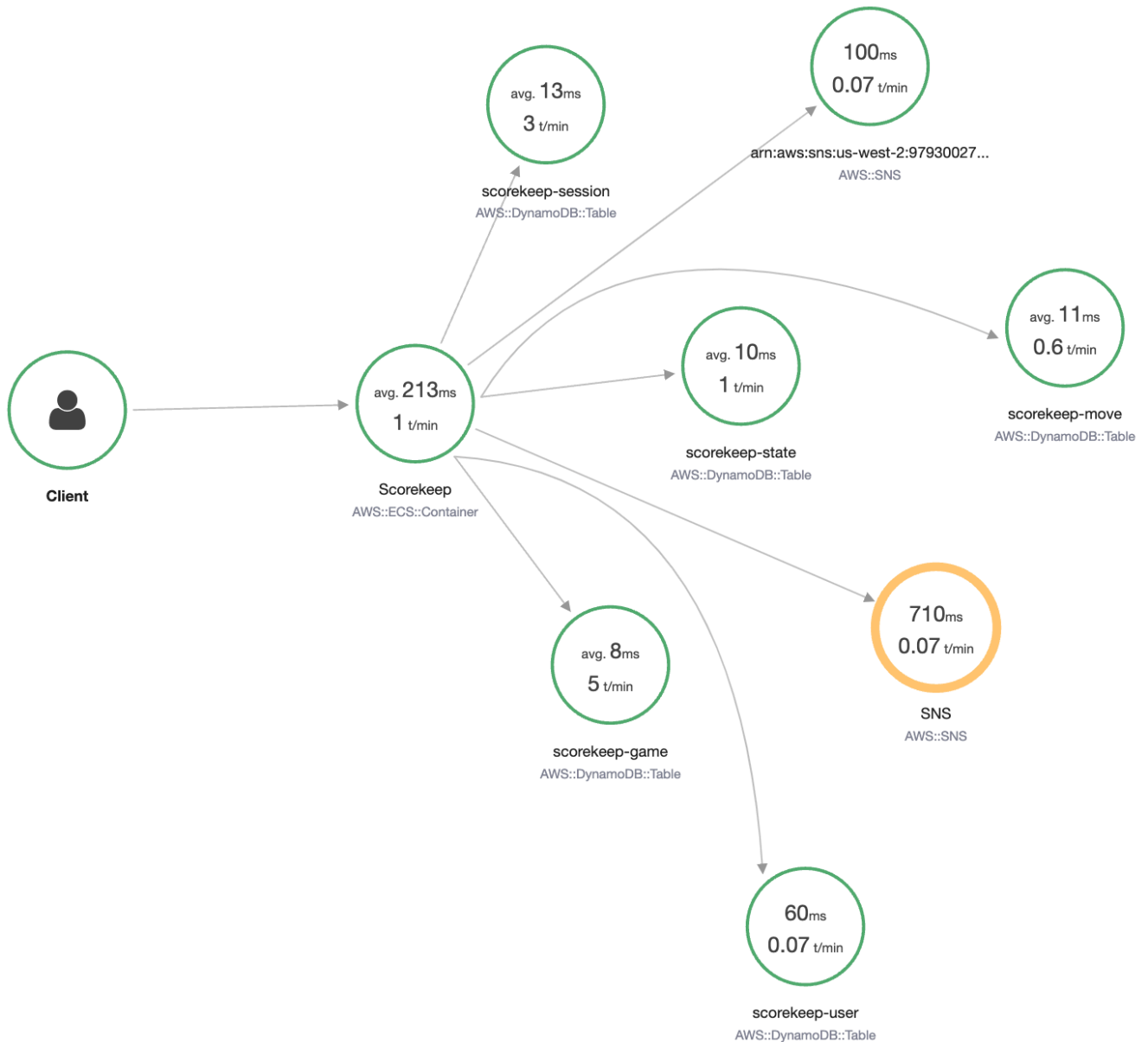
#### X-Ray 콘솔 사용하기

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 트레이스가 모두 표시되도록 하십시오.



추적 맵에는 웹 앱 클라이언트, Amazon ECS에서 실행되는 API, 애플리케이션이 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.

Trace overview

Group by:

URL ▲	Avg Latency ▲	% of Traces ▼	Response ▲
-	1.3 sec	100.00%	1 OK, 0 Throttled, 0 Errors, 0 Faults

Trace list (1)

ID ▲	Age ▼	Method ▲	Response ▲	Latency ▲	URL ▲	Client IP ▲	Annotations ▲
...48b5a191	1.1 min			1.3 sec			0

4. 페이지 하단의 Amazon SNS 세그먼트에서 오류 상태 아이콘을 선택하여 SNS 하위 세그먼트의 예외 페이지가 열립니다.

Traces > Details


Q 1-62f40175-86b347fc50bc57a992e9b835 ? ↻

Timeline
Raw data


Method	Response	Duration	Age	ID
--	--	2.1 sec	8.3 min (2022-08-10 19:05:25 UTC)	1-62f40175-86b347fc50bc57a992e9b835

▼ Trace Map


🗨 🔍 📏 🔍 Map legend ⓘ



**Client**



**Scorekeep**  
AWS::EC2::Instance



**SNS**  
AWS::SNS

Services Icons

None Health Traffic

No node resizing

Name	Res.	Duration	Status	0.0ms	200ms	400ms	600ms	800ms	1.0s	1.2s	1.4s	1.6s	1.8s	2.0s	2.2s
▼ <b>Scorekeep</b> AWS::EC2::Instance															
Scorekeep	-	2.1 sec	✓	[Timeline bar for Scorekeep]											
SNS	400	728 ms	⚠	[Timeline bar for SNS]											
▶ <b>SNS</b> AWS::SNS (Client Response)															

- X-Ray SDK는 구성된 AWS SDK 클라이언트에서 발생하는 예외를 자동으로 캡처하고 스택 추적을 기록합니다.

**Subsegment - SNS**

**Overview** | **Resources** | **Annotations** | **Metadata** | **Exceptions**

Working directory /var/app/current  
Paths --

**Cause**

com.amazonaws.services.sns.model.InvalidParameterException: Invalid parameter: Email address (Service: AmazonSNS; Status Code: 400; Error Code: InvalidParameter; Request ID: 8d29cd97-003a-5e7d-9dfb-9cffe0b7b9ab)

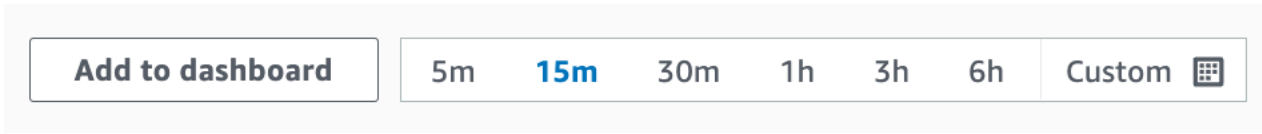
at handleErrorResponse (AmazonHttpClient.java:1545)  
at executeOneRequest (AmazonHttpClient.java:1183)  
at executeHelper (AmazonHttpClient.java:964)  
at doExecute (AmazonHttpClient.java:676)  
at executeWithTimer (AmazonHttpClient.java:650)  
at execute (AmazonHttpClient.java:633)  
at access\$300 (AmazonHttpClient.java:601)  
at execute (AmazonHttpClient.java:583)  
at execute (AmazonHttpClient.java:447)  
at doInvoke (AmazonSNSClient.java:2003)  
at invoke (AmazonSNSClient.java:1979)  
at subscribe (AmazonSNSClient.java:1881)  
at createSubscription (Utils.java:34)  
at <clinit> (WebConfig.java:52)  
at forName0 (Class.java:-2)  
at forName (Class.java:348)

**Close**

## CloudWatch console

### CloudWatch 콘솔 사용

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 트레이스가 모두 표시되도록 하십시오.



추적 맵에는 웹 앱 클라이언트, Amazon EC2에서 실행되는 API, 애플리케이션이 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.





## 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.

**Traces Info** 5m 15m **30m** 1h 3h 6h Custom

Find traces by typing a query, build a query using the Query refiners section, or [choose a sample query](#). You can also [find a trace by ID](#).

Filter by X-Ray group:

**Run query** ✔ 1 traces retrieved

---

**Query refiners**

**Traces (1)** Add to dashboard

This table shows the most recent traces with an average response time of 2.11s. It shows as many as 1000 traces.

ID	Trace status	Timestamp	Response code	Response Time	Duration
<a href="#">...86b347fc50bc57a992e9b835</a>	✔ OK	19.1min (2022-08-10 12:05:25)	-	2.11s	2.11s

4. 세그먼트 타임라인 하단에서 Amazon SNS 하위 세그먼트를 선택하고, SNS 하위 세그먼트의 예외 탭을 선택하여 예외 세부 정보를 확인합니다.

**Segments Timeline Info**

Segment status	Response code	Duration
Scorekeep AWS::EC2::Instance		
Scorekeep	✔ OK	2.11s
SNS	⊗ Fault (5xx)	728ms
SNS AWS::SNS		
SNS	⚠ Error (4xx)	728ms

---

**Segment details: SNS**

Overview | Resources | **Exceptions**

**Exceptions**

Working Directory	Paths	message
-	-	Invalid parameter: Email address (Service: AmazonSNS; Status Code: 400; Error Code: InvalidParameter; Request ID: 8b80c997-630d-5c94-a67f-92f960ba0d3e)

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 활성화하도록 알림 이메일을 구성하고 추적 맵에 강조 표시된 장애를 해결하십시오.

### AWS Management Console

를 사용하여 Amazon SNS 알림을 구성하려면 AWS Management Console

1. [CloudFormation 콘솔](#)을 엽니다
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.

EcsInstanceTypeT3

Specifies the EC2 instance type for your container instances. Defaults to t3.micro.

t3.micro

Email

UPDATE\_ME

FrontendImageUri

public.ecr.aws/xray/scorekeep-frontend:latest

5. 페이지의 하단으로 스크롤하고 다음(Next)을 선택합니다.
6. 리뷰 페이지 하단으로 스크롤하여 사용자 지정 이름으로 IAM 리소스를 생성할 CloudFormation 수 있는 확인 확인란을 선택하고 Update stack을 선택합니다.
7. 이제 CloudFormation 스택이 업데이트되고 있습니다. 스택 상태는 약 5분 동안 UPDATE\_IN\_PROGRESS로 유지되다가 UPDATE\_COMPLETE로 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` 될 때까지 기다리십시오. 몇 분 정도 걸립니다. 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. 추적 맵을 열어 SNS 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 DynamoDB 테이블, Amazon ECS 클러스터 및 ECS에서 Scorekeep을 실행하는 데 필요한 기타 서비스가 생성됩니다. 를 통해 ECS용 작업 정의 파일이 생성됩니다. CloudFormation 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 TracingFilter를 추가합니다.

Example WebConfigsrc/main/java/scorekeep/.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 단순히 AWS SDK 관련 하위 모듈을 종속성으로 사용하고, X-Ray SDK for Java는 모든 SDK 클라이언트를 자동으로 계측합니다. AWS

애플리케이션은 Docker을 사용하여 Gradle Docker Image와 함께 소스 코드를 인스턴스 내에서 빌드하고 Scorekeep API Dockerfile 파일을 사용하여 Gradle이 ENTRYPOINT에서 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, SDK 인스트루멘터 서브모듈만 있으면 AWS SDK로 이루어진 다운스트림 호출을 자동으로 계측할 수 있습니다. AWS

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 `TracingHandler`인 전역 레코더를 제공하는 `AWSXRay`라는 클래스를 제공합니다. 전역 레코더를 구성하여 수신 HTTP 호출에 대해 세그먼트를 생성하는 `AWSXRayServletFilter`를 사용자 지정할 수 있습니다. 이 샘플은 `WebConfig` 클래스에 플러그인 및 샘플링 규칙을 사용하여 전역 레코더를 구성하는 정적 블록을 포함합니다.

#### Example `WebConfig`src/main/java/scorekeep/.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 설명은 [AWS X-Ray 샘플 애플리케이션](#) 섹션을 참조하세요.

## 선택 사항: 최소 권한 정책

Scorekeep ECS 컨테이너는 AmazonSNSFullAccess 및 AmazonDynamoDBFullAccess와 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 대해 자세히 알아보려면 [AWS X-Ray의 ID 및 액세스 관리를 참조하십시오](#).

Example cf-resources.yaml 템플릿 ECS 정의 TaskRole

```
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 콘솔](#)을 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

```
{
  "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 스택을 삭제하기 전에 새 정책을 분리해야 합니다. 이렇게 연결된 정책은 CloudFormation 스택 삭제를 차단하기 때문입니다. 정책을 삭제하면 정책이 자동으로 분리될 수 있습니다.

## 사용자 지정 IAM 정책 제거

1. [IAM 콘솔\(IAM console\)](#)을 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 자습서의 이전 섹션을 사용하여 사용자 지정 정책을 만들고 연결한 경우 CloudFormation 스택을 scorekeepRole 삭제하기 전에 에서 정책을 제거해야 합니다.

### AWS Management Console

를 사용하여 샘플 애플리케이션을 삭제합니다. AWS Management Console

1. [CloudFormation 콘솔](#)을 엽니다
2. 목록에서 scorekeep 스택 이름 옆에 있는 라디오 버튼을 선택한 다음 삭제를 선택합니다.
3. 현재 CloudFormation 스택이 삭제되고 있습니다. 스택 상태는 모든 리소스가 삭제될 때까지 몇 분 동안 DELETE\_IN\_PROGRESS로 유지됩니다. 상태는 주기적으로 새로 고쳐지며, 페이지를 새로 고칠 수도 있습니다.

### AWS CLI

를 사용하여 샘플 애플리케이션을 삭제합니다. 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 5분이 소요됩니다. 다음 AWS CLI 명령을 사용하여 상태를 확인합니다.

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

## 다음 단계

다음 장 [개념](#)에서 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 SDK를 위한 Java](#)
- Node.js용 X-Ray SDK – [AWS Node.js 전용 X-Ray SDK](#)
- .NET용 X-Ray SDK – [AWS X-Ray .NET용 SDK](#)

X-Ray 데몬을 로컬에서 실행하거나 실행하려면 [AWS참조하십시오](#). [AWS X-Ray 다이어그램](#)

의 샘플 애플리케이션에 GitHub 기여하려면 [을 참조하십시오](#). [eb-java-scorekeep](#)

## SDK 클라이언트 수동 계측 AWS

Java용 X-Ray SDK는 빌드 종속성에 SDK [인스트루먼트 하위 모듈을 AWS 포함하면 AWS 모든 SDK 클라이언트를 자동으로 계측합니다](#).

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 AmazonDynamoDBClientBuilder 핸들러로 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 인스트루먼트한 SDK 클라이언트만 트레이스 맵에 나타납니다. AWS

## 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 표현식](#)에 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 ID](#)를 setUser와 함께 기록합니다. 사용자 ID는 세그먼트의 별도 필드에 기록되면 검색용으로 인덱스되지 않습니다.

Example [src/main/java/scorekeep/.java — 사용자 ID MoveController](#)

```

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 구성](#)을 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

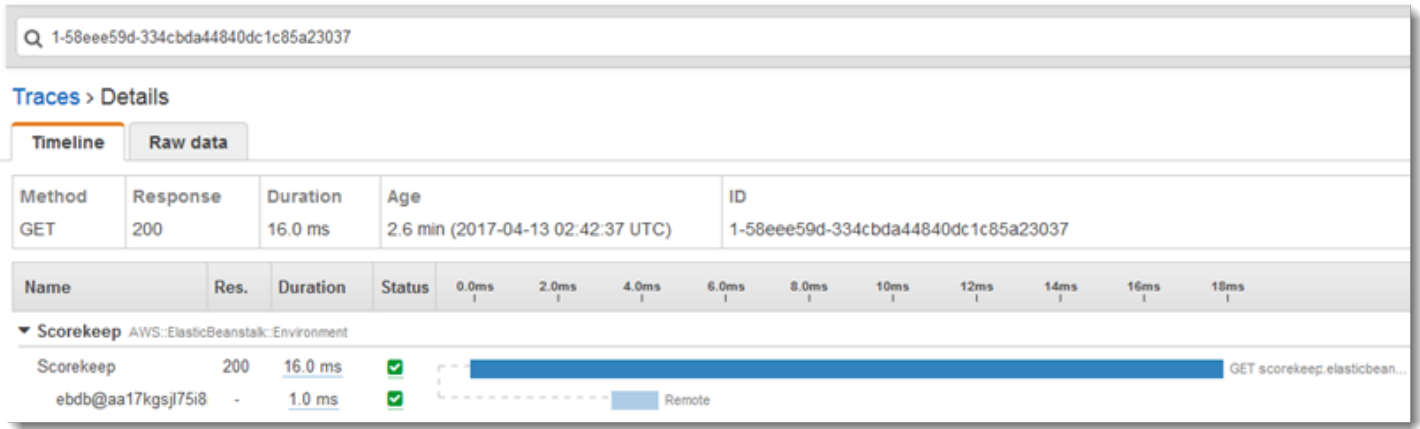
[View traces in AWS X-Ray](#)

ID	Winner	Loser
1	Mugur	Gheorghită
2	Paula	Adorján
3	Αρχίας	Stela
4	付	Pərvanə



게임 세션을 시뮬레이션하고 연결된 데이터베이스에 결과를 저장하려면 [Trace SQL queries]를 선택합니다. 그런 다음 View traces in AWS X-Ray를 선택하면 API의 /api/history 루트에 도달하는 필터링된 트레이스 목록을 볼 수 있습니다.

목록에서 트레이스 하나를 선택해 SQL 쿼리를 포함한 타임라인을 확인하십시오.



## 인스트루먼트 함수 AWS Lambda

스코어킵은 두 가지 AWS Lambda 기능을 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-name이라는 함수를 호출합니다. X-Ray SDK for Java는 계측 SDK 클라이언트를 통해 이루어진 다른 호출과 마찬가지로 Lambda에 대한 호출에 대한 정보를 하위 세그먼트에 기록합니다. AWS

### 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-worker는 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 생성할 수 있는 AWS CloudFormation 템플릿과 스크립트가 포함되어 있습니다. X-Ray SDK를 함수 코드와 함께 번들링해야 하므로 템플릿은 코드가 없는 함수를 생성합니다.

다. Scorekeep를 배포할 때 .ebextensions 폴더에 포함된 구성 파일은 SDK가 포함된 소스 번들을 생성하고 AWS Command Line Interface를 사용하여 함수 코드와 구성을 업데이트합니다.

함수

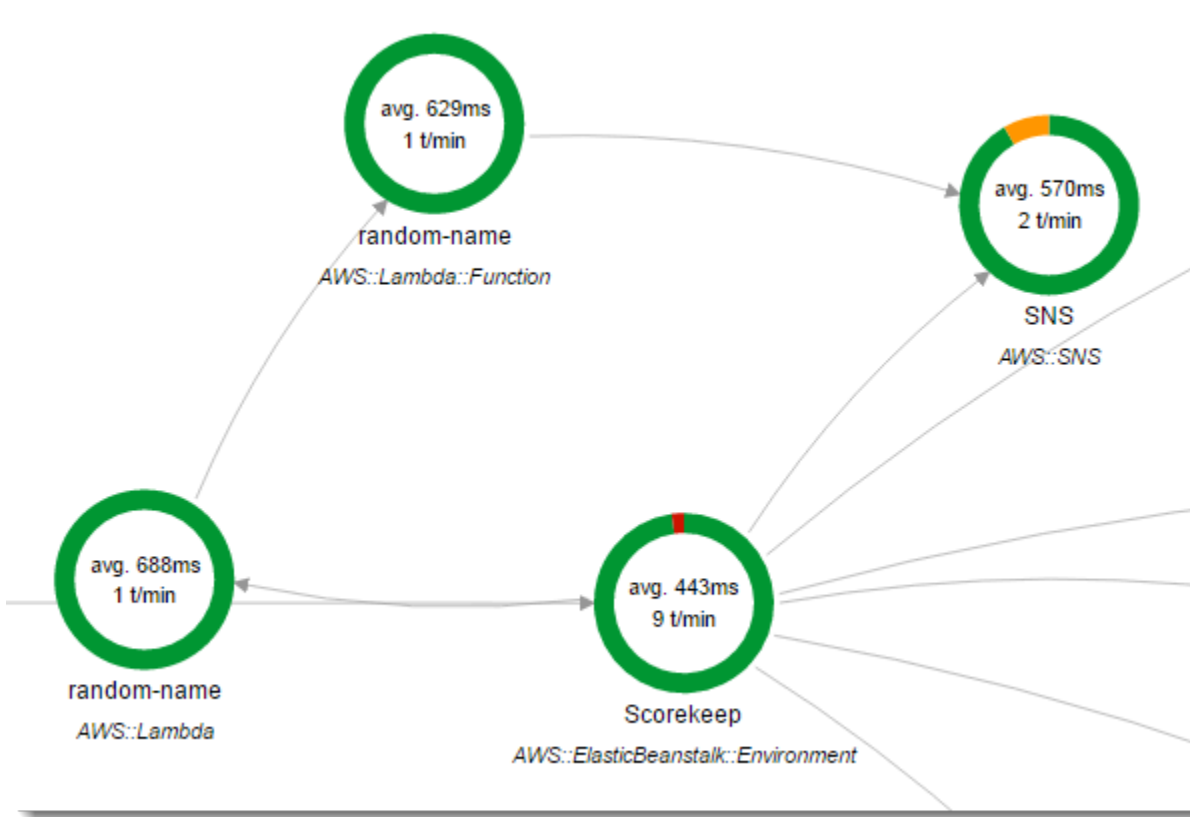
- [Random name](#)
- [작업자](#)

## Random name

사용자가 로그인하지 않거나 사용자 이름을 입력하지 않고 게임 세션을 시작하면 Scorekeep는 random name 함수를 호출합니다. Lambda는 random-name에 대한 직접 호출을 처리할 때 [추적 헤더](#)를 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 생성](#)하지 않습니다. Lambda는 세그먼트를 제공하고, 사용자는 SDK를 사용하여 클라이언트를 계측하고 하위 세그먼트를 작성합니다.



random-name 함수는 Node.js에서 구현됩니다. Node.js JavaScript 형식의 SDK를 사용하여 Amazon SNS로 알림을 보내고 Node.js 용 X-Ray SDK를 사용하여 SDK 클라이언트를 계측합니다 AWS . 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 random 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. .ebextensions폴더의 구성 파일은 배포 npm install 중에 함수 패키지를 빌드한 다음 CLI를 사용하여 Lambda 함수를 업데이트합니다.

AWS

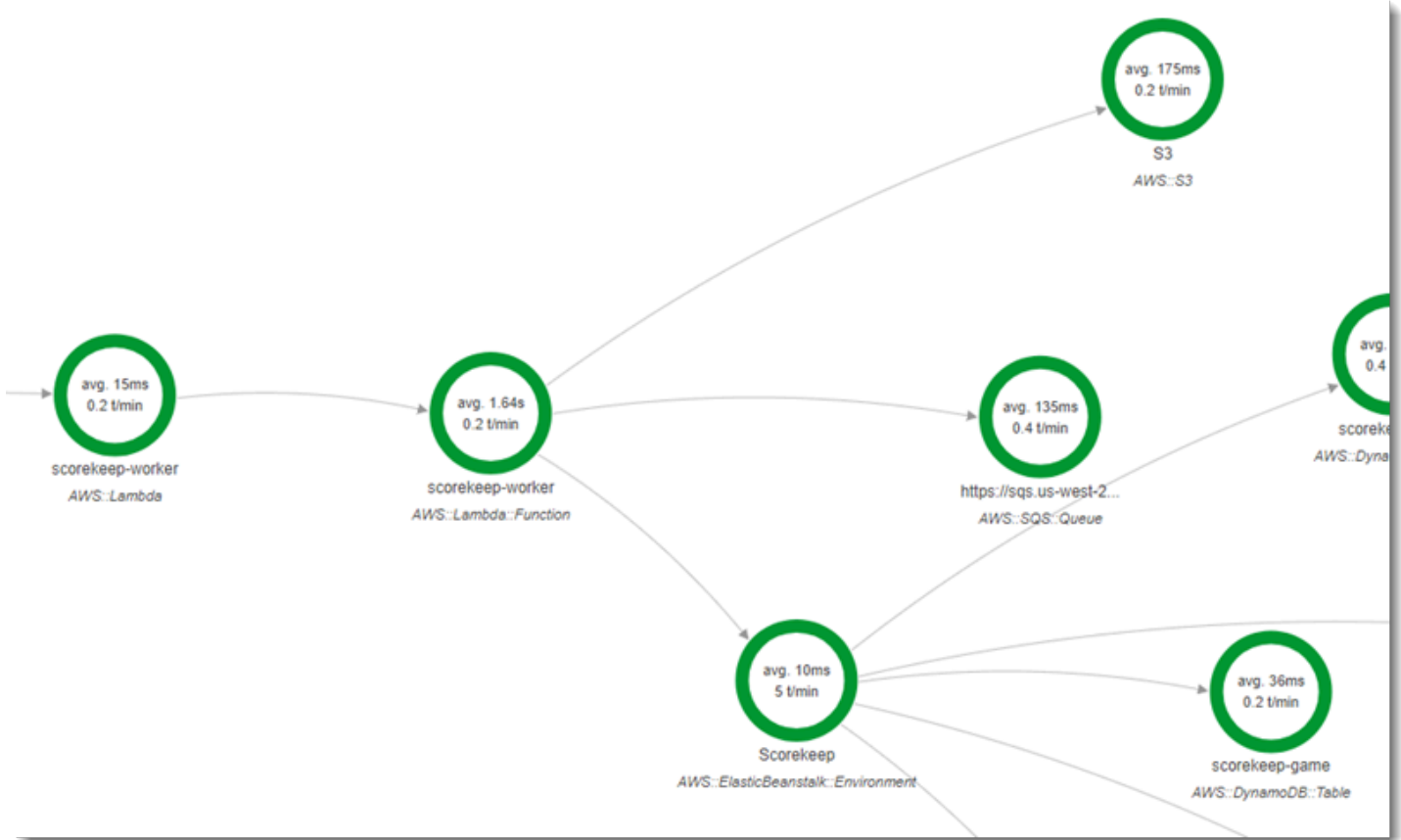
## 작업자

구성된 작업자 함수는 xray-worker라는 고유 분기에 제공됩니다. 작업자 함수와 관련 리소스를 먼저 생성하지 않은 상태에서는 실행할 수 없기 때문입니다. 지침은 [분기 readme](#)를 참조하십시오.

함수는 5분마다 번들로 제공되는 Amazon CloudWatch Events 이벤트에 의해 트리거됩니다. 이 함수가 실행되면 Scorekeep이 관리하는 Amazon SQS 대기열에서 항목을 가져옵니다. 각 메시지는 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 들어있지 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. 임의 이름 함수와 달리 워커는 인스트루먼트된 애플리케이션으로부터 요청을 받지 않으므로 추적 헤더를 수신하지 AWS Lambda 않습니다. Lambda는 활성 추적을 사용하여 추적 ID를 생성하고 샘플링 결정을 내립니다.

Python용 X-Ray SDK는 함수 맨 위에 있는 몇 줄이면 됩니다. 이 SDK를 가져와서 해당 patch\_all 함수를 실행하여 Amazon SQS 및 Amazon S3를 호출하는 데 사용하는 HTTPClient와 패치를 실행합니다. AWS SDK for Python (Boto) 이 작업자가 Scorekeep API를 호출하면 SDK는 [추적 헤더](#)를 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')

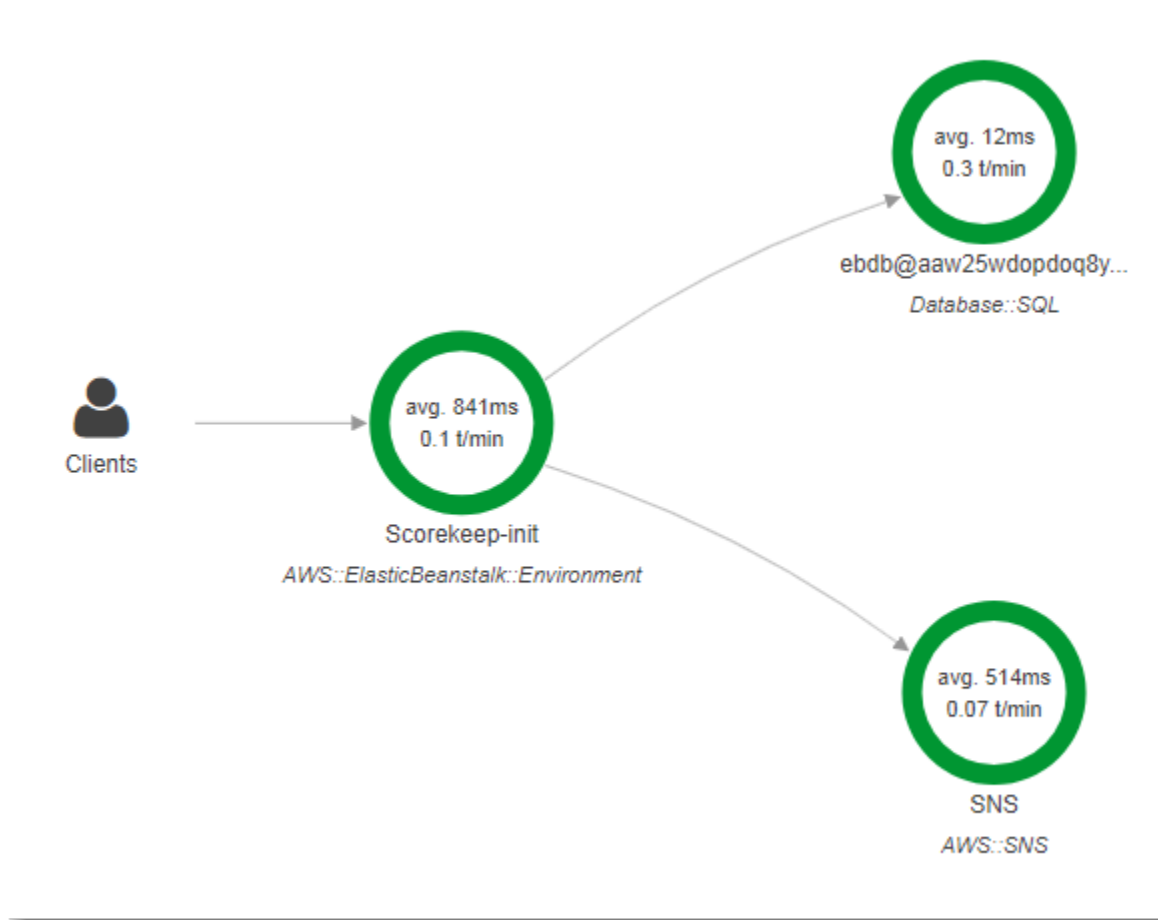
    # Receive message from SQS queue
    response = sqs.receive_message(
        QueueUrl=queue_url,
        AttributeNames=[
            'SentTimestamp'
        ],
        MaxNumberOfMessages=1,
        MessageAttributeNames=[
            'All'
        ],
        VisibilityTimeout=0,
        WaitTimeSeconds=0
    )
    ...

```

## 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 [SegmentNotFoundException](#)이 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 endSegment를 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 구성 이 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. 클라이언트가 구성되었으므로 Scorekeep이 기본 구현을 재정의하고 클라이언트가 호출될 때 SDK에서 사용할 세그먼트를 제공해야 합니다.

## 스크립트 구성

애플리케이션의 일부가 아닌 코드를 구성할 수도 있습니다. X-Ray 데몬(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.py](#)는 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 클릭하여 오류 메시지를 찾을 수 있습니다.



### Traces > Details

Timeline		Raw data		
Method	Response	Duration	Age	ID
--	--	14.6 sec	4.5 min (2017-09-14 01:25:01 UTC)	1-59b9da6d-ab8ca2666217b31a03eff86d

Name	Res.	Duration	Status	0.0ms	2.0s	4.0s	6.0s	8.0s	10s	12s	14s	16s
▼ Scorekeep-build												
Scorekeep-build	-	14.6 sec	⚠	[Progress bar showing 14.6s duration]								

Segment - Scorekeep-build
✕

Overview

Resources

Annotations

Metadata

Exceptions

Working directory	/var/app/current
Paths	/var/app/current/src/main/java/scorekeep/

**Cause**

---

```

/var/app/staging/src/main/java/scorekeep/RdsWebConfig.java:89: error: cannot find symbol
  AWSXRayRecorderBuilder builder = AWSXRayRecorderBuilder.standard().withPlugin(new EC2Plugin()).withPlugin(new ElasticBeanstalkPlugin());
                                                                    ^
symbol:   class ElasticBeanstalkPlugin
location: class RdsWebConfig
1 error

FAILURE: Build failed with an exception.
```

Close

## 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 Scorekeep 는 Amazon Cognito 자격 증명 풀을 사용하여 에서 사용할 AWS 임시 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);
    }
  })
}
```

```
}

```

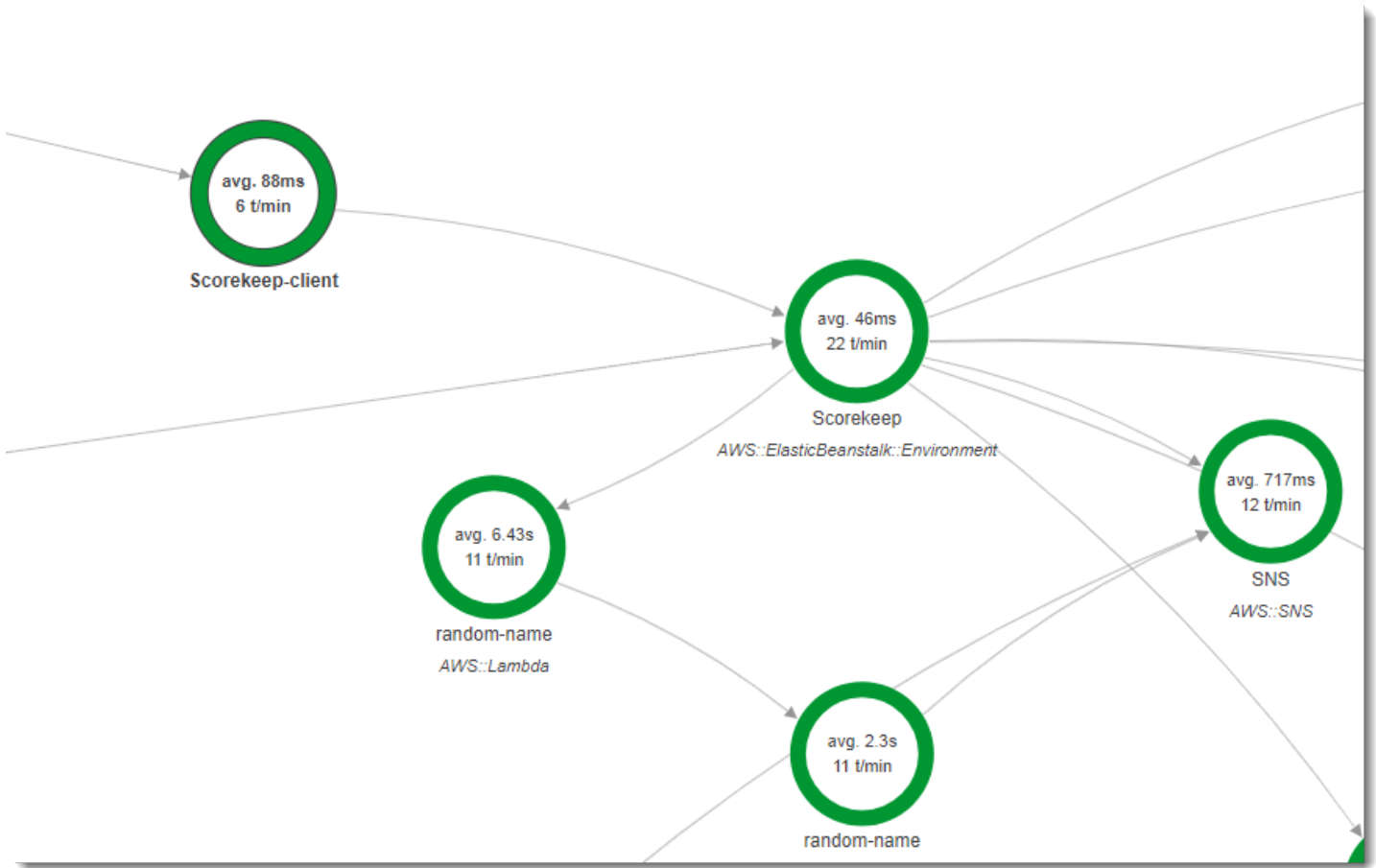
이러한 메서드는 웹 앱이 Scorekeep API를 호출하는 데 사용하는 리소스 서비스의 헤더 및 transformResponse 함수에서 호출됩니다. API가 생성하는 세그먼트와 동일한 추적에 클라이언트 세그먼트를 포함하려면 웹 앱이 [추적 헤더](#)(X-Amzn-Trace-Id)에 추적 ID와 세그먼트 ID를 포함시켜야 하며, 이 헤더는 X-Ray SDK가 읽을 수 있어야 합니다. 구성된 Java 애플리케이션이 이 헤더와 함께 요청을 수신하면 Java용 X-Ray SDK는 동일한 추적 ID를 사용하고 웹 앱 클라이언트의 세그먼트를 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);
    },
  },
  ...

```

결과 추적 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).

### Trace overview

Group by:

URL	Avg response time
<a href="http://scorekeep.elasticbeanstalk.com/#/">http://scorekeep.elasticbeanstalk.com/#/</a>	86.2 ms
<a href="http://scorekeep.elasticbeanstalk.com/#/session/4ORP7OB5/47H4SETD">http://scorekeep.elasticbeanstalk.com/#/session/4ORP7OB5/47H4SETD</a>	58.5 ms
<a href="http://scorekeep.elasticbeanstalk.com/#/game/4ORP7OB5/A94SAFFD/47H4SETD">http://scorekeep.elasticbeanstalk.com/#/game/4ORP7OB5/A94SAFFD/47H4SETD</a>	255 ms

## 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 구성된 AWS SDK for Java 클라이언트를 스레드 내에서 사용하려고 하면 `SegmentNotFoundException`이 발생하여 스레드가 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.java:
  ...
```

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.

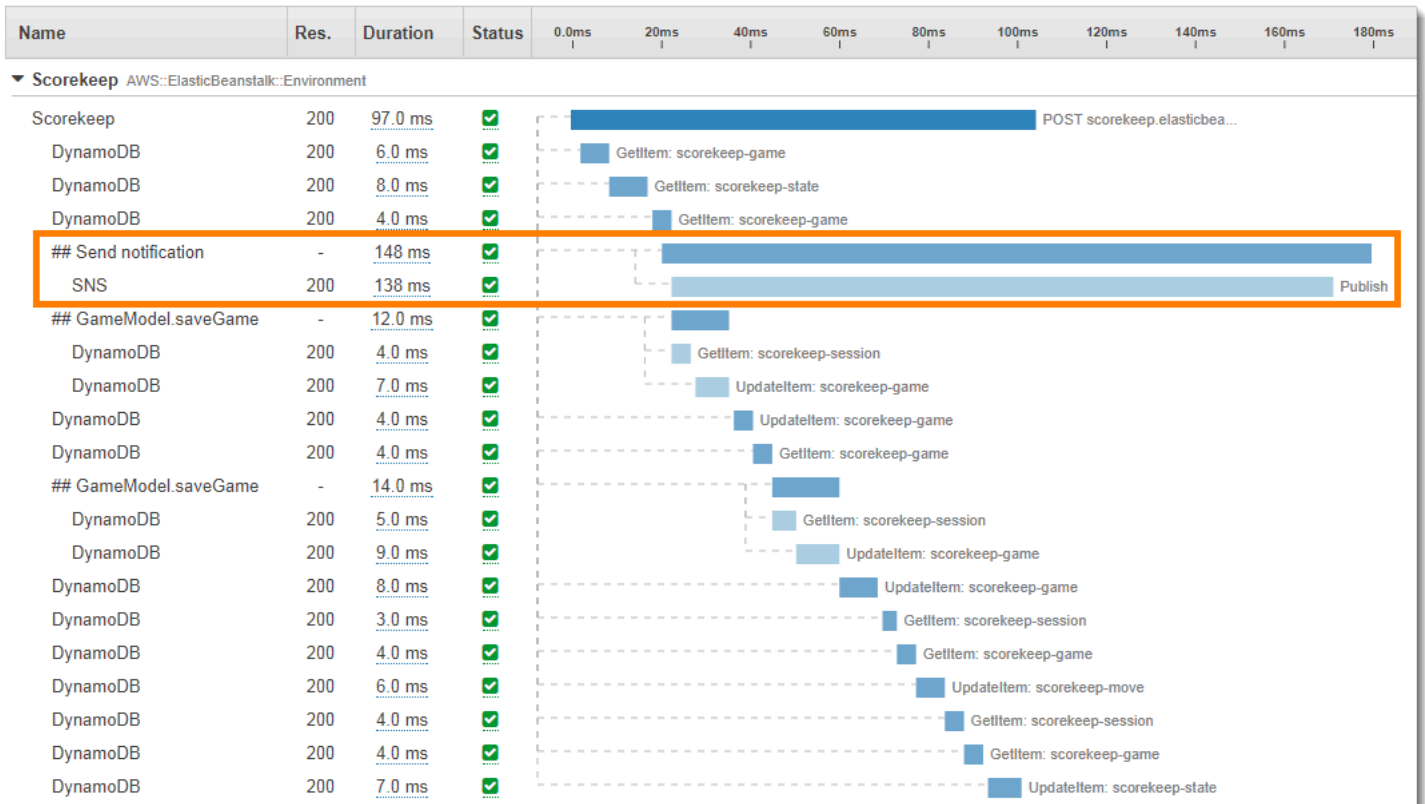
Example [src/main/java/scorekeep/MoveFactory.java](#)— 작업자 스레드에 추적 컨텍스트 전달하기

```
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.](#)

## 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](#)
- [Java용 X-Ray SDK](#)
- [Node.js용 X-Ray SDK](#)
- [X-Ray 대몬\(daemon\)](#)

## X-Ray 트레이스 맵 및 트레이스 세부 정보 페이지

다음 섹션은 X-Ray Trace 맵 및 Trace 세부 정보 페이지를 사용하는 데 문제가 있는 경우 도움이 될 수 있습니다.

### 모든 CloudWatch 로그가 보이지 않아요

X-Ray Trace 맵 및 trace 세부 정보 페이지에 표시되도록 로그를 구성하는 방법은 서비스에 따라 다릅니다.

- API Gateway에 로깅이 활성화되어 있다면 API Gateway 로그가 나타납니다.

모든 서비스 맵 노드가 관련 로그 보기를 지원하는 것은 아닙니다. 다음 노드 유형에 대한 로그를 확인하십시오.

- 람다 컨텍스트
- Lambda 함수
- API Gateway 스테이지
- Amazon ECS 클러스터
- 아마존 ECS 인스턴스
- Amazon ECS 서비스
- Amazon ECS 태스크



- Amazon EKS 클러스터
- 아마존 EKS 네임스페이스
- 아마존 EKS 노드
- 아마존 EKS 포드
- 아마존 EKS 서비스

## X-Ray 추적 맵에 모든 알람이 표시되지 않습니다.

X-Ray trace 맵에는 해당 노드와 관련된 경보가 ALARM 상태인 경우 해당 노드에 대한 경고 아이콘만 표시됩니다.

추적 맵은 다음 로직을 사용하여 경보를 노드와 연결합니다.

- 노드가 AWS 서비스를 나타내는 경우 해당 서비스와 연결된 네임스페이스를 가진 모든 경보가 노드와 연결됩니다. 예를 들어, 유형의 노드는 `AWS::Kinesis` 네임스페이스의 메트릭을 기반으로 하는 모든 경보와 연결됩니다. `CloudWatch AWS/Kinesis`
- 노드가 AWS 리소스를 나타내는 경우 해당 특정 리소스의 경보가 연결됩니다. 예를 들어 이름이 "MyTable" 인 유형의 `AWS::DynamoDB::Table` 노드는 네임스페이스가 있는 메트릭을 기반으로 `AWS/DynamoDB` 하고 차원이 로 설정된 모든 경보에 연결됩니다. `TableName MyTable`
- 노드가 이름 주위의 테두리가 파선으로 식별되는 알 수 없는 유형이면 해당 노드와 연결된 경보가 없습니다.

## 트레이스 맵에 일부 AWS 리소스가 보이지 않습니다.

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 표현됩니다. 하나는 Lambda 컨테이너용이고 다른 하나는 함수용입니다. 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

추적 맵에 노드가 너무 많습니다.

X-Ray 그룹을 사용하여 맵을 여러 맵으로 분할합니다. 자세한 내용은 [그룹에 필터 표현식 사용](#)을 참조하십시오.

## Java용 X-Ray SDK

오류: 스레드 "Thread-1"com.amazonaws.xray.exctions에서 예외가 발생했습니다.

SegmentNotFoundException: 'AmazonSNS'라는 이름의 하위 세그먼트를 시작하지 못했습니다. 세그먼트를 찾을 수 없습니다.

이 오류는 X-Ray SDK가 발신 통화를 녹음하려고 시도했지만 열린 AWS세그먼트를 찾을 수 없었음을 나타냅니다. 이는 다음과 같은 상황에서 발생할 수 있습니다.

- servlet 필터가 구성되지 않음 - X-Ray SDK는 AWSXRayServletFilter라는 필터를 사용하여 수신 요청에 대한 세그먼트를 생성합니다. 수신 요청을 계측하도록 [servlet 필터를 구성](#)합니다.
- 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 구성](#)의 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](#) 단원을 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-bluebird을 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](#)를 참조하세요.

# 보안 내부 AWS X-Ray

클라우드 AWS 보안이 최우선 과제입니다. AWS 고객은 가장 보안에 민감한 조직의 요구 사항을 충족하도록 구축된 데이터 센터 및 네트워크 아키텍처의 혜택을 누릴 수 있습니다.

보안은 기업과 기업 간의 AWS 공동 책임입니다. [공동 책임 모델](#)은 이 사항을 클라우드 내 보안 및 클라우드의 보안으로 설명합니다.

- 클라우드 보안 — AWS AWS 서비스 클라우드에서 실행되는 인프라를 보호하는 역할을 합니다 AWS 클라우드. AWS 또한 안전하게 사용할 수 있는 서비스를 제공합니다. 서드 파티 감사자는 정기적으로 [AWS 규정 준수 프로그램](#)의 일환으로 보안 효과를 테스트하고 검증합니다. X-Ray에 적용되는 규정 준수 프로그램에 대한 자세한 내용은 [규정 준수 프로그램 제공 범위 내의AWS 서비스](#)를 참조하세요.
- 클라우드에서의 보안 — AWS 서비스 사용하는 항목에 따라 책임이 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 될 수 있는 다른 방법을 사용하는 방법도 배우게 됩니다.

## 주제

- [AWS X-Ray의 데이터 보호](#)
- [ID 및 액세스 관리 대상 AWS X-Ray](#)
- [에 대한 규정 준수 검증 AWS X-Ray](#)
- [AWS X-Ray의 복원성](#)
- [AWS X-Ray의 인프라 보안](#)

## AWS X-Ray의 데이터 보호

AWS X-Ray는 항상 트레이스 및 관련 유휴 데이터를 암호화합니다. 규정 준수 또는 내부 요건 때문에 암호화 키를 감사하고 비활성화해야 하는 경우, 데이터 암호화에 AWS Key Management Service (AWS KMS) 키를 사용하도록 X-Ray를 구성할 수 있습니다.

X-Ray는 `aws/xray`라는 이름의 AWS 관리형 키를 제공합니다. [AWS CloudTrail에서 키 사용을 감사](#)하고 키 자체를 관리할 필요가 없으면 이 키를 사용하십시오. 키에 대한 액세스를 관리하거나 키 교체를 구성해야 할 경우 [고객 관리 키를 생성](#)할 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

X-Ray가 KMS 키로 추적 데이터를 암호화하거나 암호화를 해제할 때 AWS KMS 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 요금](#)을 참조하세요.

#### Note

X-Ray 인사이트 알림은 현재 고객 관리 키를 지원하지 않는 Amazon EventBridge로 이벤트를 전송합니다. 자세한 내용은 [Amazon EventBridge의 데이터 보호](#)를 참조하십시오.

고객 관리 키를 사용하여 암호화된 트레이스를 보도록 X-Ray를 구성하려면 고객 관리 키에 대한 사용자 수준 액세스 권한이 있어야 합니다. 자세한 정보는 [암호화에 대한 사용자 권한](#) 섹션을 참조하세요.

## CloudWatch console

CloudWatch 콘솔을 사용하여 암호화에 KMS 키를 사용하도록 X-Ray를 구성하려면

1. AWS Management Console에 로그인하고 <https://console.aws.amazon.com/cloudwatch/>에서 CloudWatch 콘솔을 엽니다.
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 콘솔](#)을 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 [X-Ray API를 사용하여 샘플링, 그룹 및 암호화 설정 구성](#) 섹션을 참조하십시오.

## ID 및 액세스 관리 대상 AWS X-Ray

AWS Identity and Access Management (IAM) 은 관리자가 AWS 리소스에 대한 액세스를 안전하게 제어할 수 AWS 서비스 있도록 도와줍니다. IAM 관리자는 누가 X-Ray 리소스를 사용하도록 인증되고(로그인됨) 권한이 부여되는지(권한 있음)를 제어합니다. IAM은 추가 AWS 서비스 비용 없이 사용할 수 있습니다.

### 주제

- [고객](#)
- [자격 증명을 통한 인증](#)
- [정책을 사용한 액세스 관리](#)
- [AWS X-Ray IAM과의 작동 방식](#)
- [AWS X-Ray ID 기반 정책 예제](#)
- [AWS X-Ray 자격 증명 및 액세스 문제 해결](#)

### 고객

사용 방법 AWS Identity and Access Management (IAM) 은 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 방식을 이해하면 적절한 권한을 관리자에게 요청할 수 있습니다. X-Ray의 기능에 액세스할 수 없는 경우 [AWS X-Ray 자격 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 IAM과의 작동 방식](#) 섹션을 참조하세요.

IAM 관리자 - IAM 관리자라면 X-Ray에 대한 액세스 권한 관리 정책 작성 방법을 자세히 알고 싶은 것입니다. IAM에서 사용할 수 있는 X-Ray 자격 증명 기반 정책 예제를 보려면 [AWS X-Ray ID 기반 정책 예제](#) 섹션을 참조하세요.

## 자격 증명을 통한 인증

인증은 ID 자격 증명을 AWS 사용하여 로그인하는 방법입니다. IAM 사용자로 인증 (로그인 AWS) 하거나 IAM 역할을 맡아 인증 (로그인) 해야 합니다. AWS 계정 루트 사용자

ID 소스를 통해 제공된 자격 증명을 사용하여 페더레이션 ID로 로그인할 수 있습니다. AWS IAM Identity Center (IAM ID 센터) 사용자, 회사의 싱글 사인온 인증, Google 또는 Facebook 자격 증명이 페더레이션 ID의 예입니다. 연동 자격 증명으로 로그인할 때 관리자가 이전에 IAM 역할을 사용하여 ID 페더레이션을 설정했습니다. 페더레이션을 사용하여 액세스하는 경우 AWS 간접적으로 역할을 맡게 됩니다.

사용자 유형에 따라 AWS Management Console 또는 AWS 액세스 포털에 로그인할 수 있습니다. 로그인에 대한 자세한 내용은 AWS 로그인 사용 설명서의 [내 로그인 방법을](#) 참조하십시오. AWS 계정

AWS 프로그래밍 방식으로 액세스하는 경우 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 직접 요청에 서명하는 방법에 대한 자세한 내용은 IAM 사용 설명서의 AWS [API 요청 서명을](#) 참조하십시오.

사용하는 인증 방법에 상관없이 추가 보안 정보를 제공해야 할 수도 있습니다. 예를 들어, AWS 계정의 보안을 강화하기 위해 다단계 인증 (MFA)을 사용할 것을 권장합니다. 자세한 내용은 AWS IAM Identity Center 사용 설명서의 [다중 인증](#) 및 IAM 사용자 설명서의 [AWS에서 다중 인증\(MFA\) 사용](#)을 참조하세요.

### AWS 계정 루트 사용자

계정을 AWS 계정만들 때는 먼저 계정의 모든 AWS 서비스 리소스에 대한 완전한 액세스 권한을 가진 하나의 로그인 ID로 시작합니다. 이 ID를 AWS 계정 루트 사용자라고 하며, 계정을 만들 때 사용한 이메일 주소와 비밀번호로 로그인하여 액세스할 수 있습니다. 일상적인 작업에 루트 사용자를 사용하지 않을 것을 강력히 권장합니다. 루트 사용자 보안 인증 정보를 보호하고 루트 사용자만 수행할 수 있는 작업을 수행하는 데 사용합니다. 루트 사용자로 로그인해야 하는 전체 작업 목록은 IAM 사용 설명서의 [루트 사용자 자격 증명에 필요한 태스크](#)를 참조하세요.

### IAM 사용자 및 그룹

[IAM 사용자는 단일 사용자](#) 또는 애플리케이션에 대한 특정 권한을 가진 사용자 내의 자격 증명입니다. AWS 계정 가능하면 암호 및 액세스 키와 같은 장기 자격 증명에 있는 IAM 사용자를 생성하는 대신 임시 자격 증명에 사용하는 것이 좋습니다. 하지만 IAM 사용자의 장기 자격 증명에 필요한 특정 사용 사례가 있는 경우 액세스 키를 교체하는 것이 좋습니다. 자세한 내용은 IAM 사용자 설명서의 [장기 보안 인증이 필요한 사용 사례의 경우 정기적으로 액세스 키 교체](#)를 참조하세요.



[IAM 그룹](#)은 IAM 사용자 컬렉션을 지정하는 보안 인증입니다. 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 수입할 수 있습니다. 사용자는 영구적인 장기 보안 인증 정보를 가지고 있지만, 역할은 임시 보안 인증 정보만 제공합니다. 자세한 정보는 IAM 사용자 설명서의 [IAM 사용자를 만들어야 하는 경우\(역할이 아님\)](#)를 참조하세요.

## IAM 역할

[IAM 역할](#)은 특정 권한을 가진 사용자 AWS 계정 내의 자격 증명입니다. IAM 사용자와 유사하지만, 특정 개인과 연결되지 않습니다. 역할을 AWS Management Console [전환하여](#) 에서 일시적으로 IAM 역할을 맡을 수 있습니다. AWS CLI 또는 AWS API 작업을 호출하거나 사용자 지정 URL을 사용하여 역할을 수입할 수 있습니다. 역할 사용 방법에 대한 자세한 정보는 IAM 사용자 설명서의 [IAM 역할 사용](#)을 참조하세요.

임시 보안 인증 정보가 있는 IAM 역할은 다음과 같은 상황에서 유용합니다.

- 페더레이션 사용자 액세스 - 연동 자격 증명에 권한을 부여하려면 역할을 생성하고 해당 역할의 권한을 정의합니다. 연동 자격 증명이 인증되면 역할이 연결되고 역할에 정의된 권한이 부여됩니다. 페더레이션 역할에 대한 자세한 내용은 IAM 사용자 설명서의 [Creating a role for a third-party Identity Provider](#)(서드 파티 자격 증명 공급자의 역할 만들기) 부분을 참조하세요. IAM 자격 증명 센터를 사용하는 경우 권한 집합을 구성합니다. 인증 후 아이덴티티가 액세스할 수 있는 항목을 제어하기 위해 IAM Identity Center는 권한 세트를 IAM의 역할과 연관 짓습니다. 권한 세트에 대한 자세한 내용은 AWS IAM Identity Center 사용 설명서의 [권한 세트](#) 섹션을 참조하세요.
- 임시 IAM 사용자 권한 - IAM 사용자 또는 역할은 IAM 역할을 수입하여 특정 작업에 대한 다양한 권한을 임시로 받을 수 있습니다.
- 크로스 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 AWS 서비스 경우에는 역할을 프록시로 사용하는 대신 정책을 리소스에 직접 연결할 수 있습니다. 교차 계정 액세스를 위한 역할과 리소스 기반 정책의 차이점을 알아보려면 IAM 사용자 설명서의 [IAM 역할과 리소스 기반 정책의 차이](#)를 참조하세요.
- 서비스 간 액세스 — 일부는 다른 AWS 서비스기능을 AWS 서비스 사용합니다. 예를 들어 서비스에서 직접 호출을 수행하면 일반적으로 해당 서비스는 Amazon EC2에서 애플리케이션을 실행하거나 Amazon S3에 객체를 저장합니다. 서비스는 직접적으로 호출하는 보안 주체의 권한을 사용하거나, 서비스 역할을 사용하거나, 또는 서비스 연결 역할을 사용하여 이 태스크를 수행할 수 있습니다.

- 순방향 액세스 세션 (FAS) — IAM 사용자 또는 역할을 사용하여 작업을 수행하는 경우 보안 AWS 주체로 간주됩니다. 일부 서비스를 사용하는 경우 다른 서비스에서 다른 작업을 시작하는 작업을 수행할 수 있습니다. FAS는 전화를 거는 주체의 권한을 다운스트림 AWS 서비스서비스에 AWS 서비스 요청하기 위한 요청과 결합하여 사용합니다. FAS 요청은 다른 서비스 AWS 서비스 또는 리소스와 상호 작용이 필요한 요청을 서비스가 수신한 경우에만 이루어집니다. 이 경우 두 작업을 모두 수행할 수 있는 권한이 있어야 합니다. FAS 요청 시 정책 세부 정보는 [전달 액세스 세션](#)을 참조하세요.
- 서비스 역할 - 서비스 역할은 서비스가 사용자를 대신하여 작업을 수행하기 위해 맡는 [IAM 역할](#)입니다. IAM 관리자는 IAM 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다. 자세한 정보는 IAM 사용자 설명서의 [AWS 서비스에 대한 권한을 위임할 역할 생성](#)을 참조하세요.
- 서비스 연결 역할 — 서비스 연결 역할은 연결된 서비스 역할의 한 유형입니다. AWS 서비스서비스는 사용자를 대신하여 작업을 수행하기 위해 역할을 수입할 수 있습니다. 서비스 연결 역할은 사용자에게 AWS 계정 표시되며 해당 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 API 요청을 AWS CLI 하는 애플리케이션의 임시 자격 증명을 관리할 수 있습니다. AWS 이는 EC2 인스턴스 내에 액세스 키를 저장할 때 권장되는 방법입니다. EC2 인스턴스에 AWS 역할을 할당하고 모든 애플리케이션에서 사용할 수 있게 하려면 인스턴스에 연결된 인스턴스 프로필을 생성합니다. 인스턴스 프로파일에는 역할이 포함되어 있으며 EC2 인스턴스에서 실행되는 프로그램이 임시 보안 인증 정보를 얻을 수 있습니다. 자세한 정보는 IAM 사용자 설명서의 [IAM 역할을 사용하여 Amazon EC2 인스턴스에서 실행되는 애플리케이션에 권한 부여](#)를 참조하세요.

IAM 역할을 사용할지 또는 IAM 사용자를 사용할지를 알아보려면 [IAM 사용자 설명서](#)의 IAM 역할(사용자 대신)을 생성하는 경우를 참조하세요.

## 정책을 사용한 액세스 관리

정책을 생성하고 이를 AWS ID 또는 리소스에 AWS 연결하여 액세스를 제어할 수 있습니다. 정책은 ID 또는 리소스와 연결될 때 AWS 해당 권한을 정의하는 객체입니다. AWS 주도자 (사용자, 루트 사용자 또는 역할 세션) 가 요청할 때 이러한 정책을 평가합니다. 정책의 권한이 요청 허용 또는 거부 여부를 결정합니다. 대부분의 정책은 JSON 문서로 AWS 저장됩니다. JSON 정책 문서의 구조와 콘텐츠에 대한 자세한 정보는 IAM 사용자 설명서의 [JSON 정책 개요](#)를 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, 사용자와 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 수입할 수 있습니다.

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

## 보안 인증 기반 정책

보안 인증 기반 정책은 IAM 사용자, 사용자 그룹 또는 역할과 같은 보안 인증에 연결할 수 있는 JSON 권한 정책 문서입니다. 이러한 정책은 사용자와 역할이 어떤 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 제어합니다. 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용자 설명서의 [IAM 정책 생성](#)을 참조하세요.

보안 인증 기반 정책은 인라인 정책 또는 관리형 정책으로 한층 더 분류할 수 있습니다. 인라인 정책은 단일 사용자, 그룹 또는 역할에 직접 포함됩니다. 관리형 정책은 내 여러 사용자, 그룹 및 역할에 연결할 수 있는 독립형 정책입니다. AWS 계정관리형 정책에는 AWS 관리형 정책과 고객 관리형 정책이 포함됩니다. 관리형 정책 또는 인라인 정책을 선택하는 방법을 알아보려면 IAM 사용자 설명서의 [관리형 정책과 인라인 정책의 선택](#)을 참조하세요.

## 리소스 기반 정책

리소스 기반 정책은 리소스에 연결하는 JSON 정책 설명서입니다. 리소스 기반 정책의 예는 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 보안 주체가 해당 리소스와 어떤 조건에서 어떤 태스크를 수행할 수 있는지를 정의합니다. 리소스 기반 정책에서 [보안 주체를 지정](#)해야 합니다. 보안 주체에는 계정, 사용자, 역할, 연동 사용자 등이 포함될 수 있습니다. AWS 서비스

리소스 기반 정책은 해당 서비스에 있는 인라인 정책입니다. IAM의 AWS 관리형 정책은 리소스 기반 정책에 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 설명서 형식을 사용하지 않지만 리소스 기반 정책과 유사합니다.

ACL을 지원하는 서비스의 예로는 아마존 S3와 아마존 VPC가 있습니다. AWS WAF ACL에 대해 자세히 알아보려면 Amazon Simple Storage Service 개발자 안내서의 [ACL\(액세스 제어 목록\) 개요](#) 섹션을 참조하세요.

## 기타 정책 타입

AWS 일반적이지 않은 추가 정책 유형을 지원합니다. 이러한 정책 타입은 더 일반적인 정책 타입에 따라 사용자에게 부여되는 최대 권한을 설정할 수 있습니다.

- 권한 경계 – 권한 경계는 자격 증명 기반 정책에 따라 IAM 엔티티(IAM 사용자 또는 역할)에 부여할 수 있는 최대 권한을 설정하는 고급 특성입니다. 개체에 대한 권한 경계를 설정할 수 있습니다. 그 결과로 얻는 권한은 개체의 보안 인증 기반 정책과 그 권한 경계의 교집합입니다. Principal 필드에서 사용자나 역할을 보안 주체로 지정하는 리소스 기반 정책은 권한 경계를 통해 제한되지 않습니다. 이러한 정책 중 하나에 포함된 명시적 거부는 허용을 재정의합니다. 권한 경계에 대한 자세한 정보는 IAM 사용자 설명서의 [IAM 엔티티에 대한 권한 경계](#)를 참조하세요.
- 서비스 제어 정책 (SCP) - SCP는 조직 또는 조직 단위 (OU)에 대한 최대 권한을 지정하는 JSON 정책입니다. AWS Organizations AWS Organizations 사업체가 소유한 여러 AWS 계정 개를 그룹화하고 중앙에서 관리하는 서비스입니다. 조직에서 모든 특성을 활성화할 경우 서비스 제어 정책 (SCP)을 임의의 또는 모든 계정에 적용할 수 있습니다. SCP는 각 항목을 포함하여 구성원 계정의 엔티티에 대한 권한을 제한합니다. AWS 계정 루트 사용자조직 및 SCP에 대한 자세한 정보는 AWS Organizations 사용 설명서의 [SCP 작동 방식](#)을 참조하세요.
- 세션 정책 – 세션 정책은 역할 또는 페더레이션 사용자에게 대해 임시 세션을 프로그래밍 방식으로 생성할 때 파라미터로 전달하는 고급 정책입니다. 결과적으로 얻는 세션의 권한은 사용자 또는 역할의 자격 증명 기반 정책의 교집합과 세션 정책입니다. 또한 권한을 리소스 기반 정책에서 가져올 수도 있습니다. 이러한 정책 중 하나에 포함된 명시적 거부는 허용을 재정의합니다. 자세한 정보는 IAM 사용자 설명서의 [세션 정책](#)을 참조하세요.

## 여러 정책 타입

여러 정책 타입이 요청에 적용되는 경우 결과 권한은 이해하기가 더 복잡합니다. 여러 정책 유형이 관련된 경우 요청을 허용할지 여부를 AWS 결정하는 방법을 알아보려면 IAM 사용 설명서의 [정책 평가 로직](#)을 참조하십시오.

## AWS X-Ray IAM과의 작동 방식

IAM을 사용하여 X-Ray에 대한 액세스를 관리하려면 먼저 어떤 IAM 기능을 X-Ray와 함께 사용할 수 있는지를 이해해야 합니다. X-Ray 및 기타 제품이 IAM과 어떻게 AWS 서비스 연동되는지 자세히 알아보려면 IAM 사용 설명서의 [IAM과AWS 서비스 연동하는](#) 방법을 참조하십시오.

AWS Identity and Access Management (IAM) 을 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) 와 상관없이 권한을 균일하게 적용합니다. AWS CLI

[X-Ray 콘솔을 사용하여](#) 추적 맵과 세그먼트를 보려면 읽기 권한만 필요합니다. 콘솔 액세스를 활성화하려면 AWSXrayReadOnlyAccess [관리형 정책](#)을 IAM 사용자에게 추가합니다.

[로컬 개발 및 테스트](#)를 위해서는 읽기 및 쓰기 권한을 갖는 IAM 역할을 생성합니다. [역할을 맡고](#) 역할에 대한 임시 자격 증명을 저장합니다. 이러한 자격 증명은 X-Ray 데몬 AWS CLI, 및 SDK와 함께 사용할 수 있습니다. AWS 자세한 내용은 [AWS CLI임시 보안 자격 증명 사용](#)을 참조하세요.

[계측 앱을 배포하려면 AWS쓰기 권한이 있는 IAM 역할을 생성하고 애플리케이션을 실행하는 리소스에 할당하십시오.](#) AWSXRayDaemonWriteAccess트레이스를 업로드할 수 있는 권한과 샘플링 규칙 사용을 지원하는 일부 읽기 권한을 포함합니다. 자세한 정보는 [샘플링 규칙을 구성합니다.](#)을 참조하세요.

읽기 및 쓰기 정책에는 [암호화 키 설정](#) 및 샘플링 규칙을 구성할 권한은 포함되지 않습니다.

AWSXrayFullAccess을 사용하여 이러한 설정에 액세스하거나, 사용자 지정 정책에서 [구성 API](#)를 추가합니다. 생성하는 고객 관리형 키를 사용한 암호화 및 해독을 위해서는 [해당 키를 사용할 권한](#)도 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 자격 증명 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 정책에서 사용하는 모든 요소에 대해 알고 싶다면 IAM 사용 설명서의 [IAM JSON 정책 요소 참조](#)를 참조하세요.

## 작업

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.

JSON 정책의 Action요소는 정책에서 액세스를 허용하거나 거부하는 데 사용할 수 있는 태스크를 설명합니다. 정책 작업은 일반적으로 관련 AWS API 작업과 이름이 같습니다. 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 포함하십시오.

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, Get라는 단어로 시작하는 모든 태스크를 지정하려면 다음 태스크를 포함합니다.

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

X-Ray 작업 목록을 보려면 IAM 사용 설명서의 [AWS X-Ray에서 정의한 작업](#)을 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 있는지 지정할 수 있습니다.

Resource JSON 정책 요소는 작업이 적용되는 하나 이상의 개체를 지정합니다. 문장에는 Resource 또는 NotResource요소가 반드시 추가되어야 합니다. 모범 사례에 따라 [Amazon 리소스 이름\(ARN\)](#)을 사용하여 리소스를 지정합니다. 리소스 수준 권한이라고 하는 특정 리소스 타입을 지원하는 작업에 대해 이 작업을 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 작업](#)이 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.

```
{
  "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.

```

{
  "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.

X-Ray 리소스 유형 및 해당 ARN의 목록을 보려면 IAM 사용 설명서의 [AWS X-Ray에서 정의된 리소스](#)를 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 글로벌 조건 키를 보려면 IAM 사용 설명서의 [AWS 글로벌 조건 컨텍스트 키](#)를 참조하십시오.

#### 예제

X-Ray 자격 증명 기반 정책의 예를 보려면 [AWS X-Ray ID 기반 정책 예제](#) 단원을 참조하세요.



## X-Ray 리소스 기반 정책

X-Ray는 [Amazon SNS 활성 추적](#)과 같은 현재 및 미래의 AWS 서비스 통합을 위한 리소스 기반 정책을 지원합니다. X-Ray 리소스 기반 정책은 다른 AWS Management Console 사용자가 업데이트하거나 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" } } } ] }'
```

이 예제를 사용하려면, *partitionregionaccount-id*, 를 특정 AWS 파티션, 지역, 계정 ID, Amazon SNS 주제 이름으로 바꾸십시오. *topic-name* 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. 태그를 기반으로 액세스를 제어하려면 `xray:ResourceTag/key-name`, `aws:RequestTag/key-name` 또는 `aws:TagKeys` 조건 키를 사용하여 정책의 [조건 요소](#)에 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 예제는 [태그에 기반한 X-Ray 그룹 및 샘플링 규칙에 대한 액세스 관리](#) 섹션에서 확인할 수 있습니다.

## 로컬에서 애플리케이션 실행

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

[데몬\(daemon\)을 로컬에서 실행하는](#) 경우 IAM 역할을 생성하고 [역할을 수입한 다음](#) 환경 변수나 사용자 폴더의 `.aws`라는 폴더 내 `credentials`라는 파일에 임시 자격 증명을 저장합니다. 자세한 내용은 [AWS CLI 임시 보안 자격 증명 사용](#)을 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 또는 AWS CLI에서 사용할 자격 증명을 이미 구성한 경우 데몬은 해당 자격 증명을 사용할 수 있습니다. 여러 프로파일을 사용할 수 있을 경우 데몬(daemon)은 기본 프로파일을 사용합니다.

## 에서 애플리케이션 실행 AWS

에서 AWS 애플리케이션을 실행할 때는 역할을 사용하여 데몬을 실행하는 Amazon EC2 인스턴스 또는 Lambda 함수에 권한을 부여합니다.

- Amazon Elastic Compute Cloud(Amazon EC2) – IAM 역할을 생성하고 [인스턴스 프로파일](#)로 EC2 인스턴스에 연결합니다.
- Amazon Elastic Container Service(Amazon ECS) — IAM 역할을 생성하고 컨테이너 인스턴스에 [컨테이너 인스턴스 IAM 역할](#)로 연결합니다.
- AWS Elastic Beanstalk (Elastic Beanstalk) — Elastic [Beanstalk는 기본 인스턴스 프로필에 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 콘솔\(IAM console\)](#)을 엽니다.
2. Roles를 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 데몬에 추적 데이터를 업로드할 수 있는 권한을 부여합니다.

애플리케이션에서 AWS SDK를 사용하여 다른 서비스에 액세스하는 경우 해당 서비스에 대한 액세스 권한을 부여하는 정책을 추가하세요.

7. 다음 단계를 선택합니다.
8. Create Role(역할 생성)을 선택합니다.

## 암호화에 대한 사용자 권한

X-Ray는 모든 트레이스 데이터를 기본적으로 암호화하며, [관리하는 키를 사용하도록 구성](#)할 수 있습니다. AWS Key Management Service 고객 관리 키를 선택하는 경우 키의 액세스 정책을 통해 X-Ray

에 암호화에 사용할 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 권한은 필요하지 않습니다. 자세한 내용은 [AWS KMS 개발자 안내서의 키 정책 사용을 참조하십시오](#).

기본 암호화의 경우 또는 AWS 관리형 CMK (`aws/xray`)를 선택하는 경우 권한은 X-Ray API에 액세스할 수 있는 사용자를 기반으로 합니다. `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 ID 기반 정책 예제

기본적으로 사용자 및 역할에는 X-Ray 리소스를 생성하거나 수정할 수 있는 권한이 없습니다. 또한 AWS Management Console, AWS CLI, 또는 AWS API를 사용하여 작업을 수행할 수 없습니다. 관리자는 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 생성](#)을 참조하십시오.

### 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 정책으로 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. 자격 증명 기반 정책을 생성하거나 편집할 때는 다음 지침과 권장 사항을 따르십시오.

- AWS 관리형 정책으로 시작하여 최소 권한 권한으로 이동 — 사용자와 워크로드에 권한을 부여하려면 여러 일반 사용 사례에 권한을 부여하는 AWS 관리형 정책을 사용하세요. 해당 내용은 에서 사용할 수 있습니다. AWS 계정사용 사례에 맞는 AWS 고객 관리형 정책을 정의하여 권한을 더 줄이는 것이 좋습니다. 자세한 정보는 IAM 사용 설명서의 [AWS 관리형 정책](#) 또는 [AWS 직무에 대한 관리형 정책](#)을 참조하세요.
- 최소 권한 적용 – IAM 정책을 사용하여 권한을 설정하는 경우 태스크를 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 권한](#)을 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. 예를 AWS 서비스들어 특정 작업을 통해 서비스 작업을 사용하는 경우 조건을 사용하여 서비스 작업에 대한 액세스 권한을 부여할 수도 AWS CloudFormation있습니다. 자세한 정보는 IAM 사용 설명서의 [IAM JSON 정책 요소: 조건](#)을 참조하세요.
- IAM Access Analyzer를 통해 IAM 정책을 검증하여 안전하고 기능적인 권한 보장 – IAM Access Analyzer에서는 IAM 정책 언어(JSON)와 모범 사례가 정책에서 준수되도록 신규 및 기존 정책을 검증합니다. IAM Access Analyzer는 100개 이상의 정책 확인 항목과 실행 가능한 추천을 제공하여 안전하고 기능적인 정책을 작성하도록 돕습니다. 자세한 정보는 IAM 사용 설명서의 [IAM Access Analyzer 정책 검증](#)을 참조하세요.
- 멀티 팩터 인증 (MFA) 필요 - IAM 사용자 또는 루트 사용자가 필요한 시나리오가 있는 경우 추가 보안을 위해 AWS 계정 MFA를 활성화하십시오. API 작업을 직접 호출할 때 MFA가 필요하다면 정책에 MFA 조건을 추가합니다. 자세한 정보는 IAM 사용 설명서의 [MFA 보호 API 액세스 구성](#)을 참조하세요.

IAM의 모범 사례에 대한 자세한 내용은 IAM 사용 설명서의 [IAM의 보안 모범 사례](#)를 참조하세요.

## X-Ray 콘솔 사용하기

AWS X-Ray 콘솔에 액세스하려면 최소한의 권한이 있어야 합니다. 이러한 권한을 통해 사용자의 X-Ray 리소스에 대한 세부 정보를 나열하고 볼 수 있어야 AWS 계정입니다. 최소 필수 권한보다 더 제한적인 자격 증명 기반 정책을 만들면 콘솔이 해당 정책에 연결된 엔터티(사용자 또는 역할)에 대해 의도대로 작동하지 않습니다.

이러한 개체가 여전히 X-Ray 콘솔을 사용할 수 있도록 하려면 `AWSXRayReadOnlyAccess` AWS 관리형 정책을 개체에 연결하십시오. 이 정책은 [X-Ray의 IAM 관리형 정책](#)에 자세히 설명되어 있습니다. 자세한 내용은 IAM 사용 설명서의 [사용자에게 권한 추가](#)를 참조하십시오.

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

### 사용자가 자신이 권한을 볼 수 있도록 허용

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

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ViewOwnUserInfo",
      "Effect": "Allow",
      "Action": [
        "iam:GetUserPolicy",
        "iam:ListGroupsWithUser",
        "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 지정하기](#)를 참조하십시오.

태그를 `stage:prod` 또는 `stage:preprod`를 사용하여 그룹을 생성, 업데이트 또는 삭제할 수 있는 사용자 액세스를 거부하려면 다음과 유사한 정책을 사용하여 사용자에게 역할을 할당하십시오.

```

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

```

```

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

```

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.

```

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

```



```

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

```

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=preprod 또는 stage=prod 태그를 지정해서는 안 됩니다. 조건 키 이름은 대소문자를 구분하지 않기 때문에 조건 태그 키 Stage는 Stage 및 stage 모두와 일치합니다. 조건 블록에 대한 자세한 내용은 IAM 사용 설명서의 [IAM JSON 정책 요소: 조건](#)을 참조하세요.

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

```

{
  "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 API를 출시할 때 새 권한으로 이러한 관리형 정책을 업데이트할 수 있습니다. AWS X-Ray 읽기 전용, 쓰기 전용, 관리자 사용 사례에 대한 관리형 정책을 제공합니다.

- **AWSXrayReadOnlyAccess**— X-Ray 콘솔 또는 AWS SDK를 사용하여 X-Ray API에서 추적 데이터, 추적 맵, 통찰력 및 X-Ray 구성을 가져오기 위한 읽기 권한. [AWS CLIOAM \(Observability Access Manager\) oam:ListSinks](#) 및 [계정 간 업저버빌리티의 일환으로 소스 계정에서 공유된 트레이스를 콘솔에서 볼 수 있도록 하는 oam:ListAttachedSinks](#) 권한이 포함됩니다. [CloudWatch BatchGetTraceSummaryById](#) 및 [GetDistinctTraceGraphs](#) API 작업은 코드를 통해 호출되지 않으며, 및 SDK에 포함되지 않습니다. AWS CLI `AWS`

```

{
  "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 또는 AWS SDK를 사용하여 세그먼트 문서 및 원격 분석을 X-Ray API에 업로드하기 위한 쓰기 권한. AWS CLI 샘플링 규칙을 가져오고 샘플링 결과를 보고할 읽기 권한이 포함됩니다. 자세한 정보는 [샘플링 규칙을 구성합니다](#)를 참조하세요.

```
{
  "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 [CloudWatch 계정 간의 계정 간 가시성](#)을 지원하는 데 사용됩니다.

```
{
  "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. [OAM \(Observability Access Manager\)](#) [oam:ListSinks](#) 과 [크로스 계정 업저버빌리티의 일환으로 소스 계정에서 공유된 트레이스를 콘솔에서 볼 수 있는 oam:ListAttachedSinks](#) 권한을 포함합니다. [CloudWatch](#)

```

{
  "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 콘솔\(IAM console\)](#)을 엽니다.
2. 인스턴스 프로파일, IAM 사용자 또는 IAM 사용자 그룹과 연결된 역할을 엽니다.
3. 권한에서 관리형 정책을 연결합니다.

## AWS 관리형 정책으로 X-Ray 업데이트

이 서비스가 이러한 변경 사항을 추적하기 시작한 이후 X-Ray의 AWS 관리형 정책 업데이트에 대한 세부 정보를 확인하십시오. 이 페이지의 변경 사항에 대한 자동 알림을 받으려면 X-Ray [문서 기록](#) 페이지에서 RSS 피드를 구독하세요.

변경 사항	설명	날짜
<a href="#">X-Ray용 IAM 관리형 정책 — 새로운 AWSXrayCrossAccountSharingConfiguration 및 업데이트된 AWSXrayReadOnlyAccess 및 AWSXrayFullAccess 정책</a> 이 추가되었습니다.	X-Ray는 이러한 정책에 Observability Access Manager (OAM) 권한을 oam:ListSinks 추가하여 콘솔에서 계정 <a href="#">CloudWatch 간</a> 관찰 가능성의 일환으로 소스 계정에서 공유된 트레이스를 볼 수 있도록 했습니다. oam:ListAttachedSinks	2022년 11월 27일
<a href="#">X-Ray용 IAM 관리형 정책 — AWSXrayReadOnlyAccess 정책 업데이트</a>	X-Ray가 ListResourcePolicies API 작업을 추가했습니다.	2022년 11월 15일
<a href="#">X-Ray 콘솔 사용 — AWSXrayReadOnlyAccess 정책 업데이트</a>	X-Ray는 두 개의 새로운 API 작업 BatchGetTraceSummaryById 과 GetDistin	2022년 11월 11일

변경 사항	설명	날짜
	<p>ctTraceGraphs 를 추가했으며,</p> <p>이 작업들은 코드로 호출되는 것이 아닙니다. 따라서 이러한 API 작업은 및 SDK에 포함되지 않습니다. AWS CLI AWS</p>	

## 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 작업](#)이 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.

```
{
  "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.

```
{
  "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.



## AWS X-Ray 자격 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 AWS 계정 외부의 사람이 내 X-Ray 리소스에 액세스할 수 있게 허용하기를 원합니다.](#)

### X-Ray에서 작업을 수행할 권한이 없음

AWS Management Console에서 태스크를 수행할 권한이 없다는 메시지가 나타나는 경우 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 서비스에서는 새 서비스 역할 또는 서비스 연결 역할을 생성하는 대신, 해당 서비스에 기존 역할을 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 서비스에 서비스 역할이 부여한 권한이 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 보안 인증 정보를 제공한 사람입니다.

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

다른 사용자가 X-Ray에 액세스하도록 허용하려면 액세스 권한이 필요한 사용자나 애플리케이션에 대한 IAM 엔터티(사용자 또는 역할)를 생성해야 합니다. 다른 사용자들은 해당 엔터티에 대한 자격 증명을 사용해 AWS에 액세스합니다. 그런 다음 X-ray에 대한 올바른 권한을 부여하는 정책을 엔터티에 연결해야 합니다.

바로 시작하려면 IAM 사용 설명서의 [첫 번째 IAM 위임 사용자 및 그룹 생성](#)을 참조하세요.

내 AWS 계정 외부의 사람이 내 X-Ray 리소스에 액세스할 수 있게 허용하기를 원합니다.

다른 계정의 사용자 또는 조직 외부의 사람이 리소스에 액세스하는 데 사용할 수 있는 역할을 생성할 수 있습니다. 역할을 수입할 신뢰할 수 있는 사람을 지정할 수 있습니다. 리소스 기반 정책 또는 ACL(액세스 제어 목록)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 IAM과의 작동 방식](#) 단원을 참조하세요.
- 소유하고 있는 AWS 계정의 리소스에 대한 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 [자신이 소유한 다른 AWS 계정의 IAM 사용자에게 대한 액세스 권한 제공](#)을 참조하세요.
- 리소스에 대한 액세스 권한을 서드 파티 AWS 계정에게 제공하는 방법을 알아보려면 IAM 사용 설명서의 [서드 파티가 소유한 AWS 계정에 대한 액세스 제공](#)을 참조하세요.
- 자격 증명 연동을 통해 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 [외부에서 인증된 사용자에게 액세스 권한 제공\(자격 증명 연동\)](#)을 참조하세요.
- 교차 계정 액세스를 위한 역할과 리소스 기반 정책 사용의 차이점을 알아보려면 IAM 사용 설명서의 [IAM 역할과 리소스 기반 정책의 차이](#)를 참조하세요.

## AWS X-Ray의 로깅 및 모니터링

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

### AWS CloudTrail 로그

AWS X-Ray는 X-Ray에서 사용자, 역할 또는 AWS 서비스가 수행한 API 작업을 기록하기 위하여 AWS CloudTrail과 통합됩니다. CloudTrail을 사용하여 X-Ray API 요청을 실시간으로 모니터링하고 Amazon S3, Amazon CloudWatch Logs 및 Amazon CloudWatch Events에 로그를 저장할 수 있습니다. 자세한 내용은 [클라우드 트레일을 사용하여 X-Ray API 호출 로깅 AWS CloudTrail](#) 섹션을 참조하세요.

### AWS Config 추적

AWS X-Ray는 AWS Config와 통합되어 X-Ray 암호화 리소스에 대한 구성 변경을 기록합니다. AWS Config를 사용하여 암호화 리소스를 인벤토리하고, X-Ray 구성 기록을 감사하고, 리소스 변경에 대한 알림을 보낼 수 있습니다. 자세한 내용은 [AWS Config를 사용하여 X-Ray 암호화 구성 변경 추적하기](#) 섹션을 참조하세요.

### 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 SDK의 메트릭스](#) 섹션을 참조하세요.

## 에 대한 규정 준수 검증 AWS X-Ray

특정 규정 준수 프로그램의 범위 내에 AWS 서비스 있는지 알아보려면 AWS 서비스 규정 준수 [프로그램의 AWS 서비스 범위별, 규정](#) 참조하여 관심 있는 규정 준수 프로그램을 선택하십시오. 일반 정보는 [AWS 규정 준수 프로그램 AWS 보증 프로그램 규정 AWS](#) 참조하십시오.

클라우드 트레일을 사용하여 AWS Artifact 타사 감사 보고서를 다운로드할 수 있습니다. 자세한 내용은 의 보고서 <https://docs.aws.amazon.com/artifact/latest/ug/downloading-documents.html> 참조하십시오 AWS Artifact.

사용 시 규정 준수 AWS 서비스 책임은 데이터의 민감도, 회사의 규정 준수 목표, 관련 법률 및 규정에 따라 결정됩니다. AWS 규정 준수에 도움이 되는 다음 리소스를 제공합니다.

- [보안 및 규정 준수 킷스타트 가이드](#) - 이 배포 가이드에서는 아키텍처 고려 사항을 설명하고 보안 및 규정 준수에 AWS 중점을 둔 기본 환경을 배포하기 위한 단계를 제공합니다.
- [Amazon Web Services의 HIPAA 보안 및 규정 준수를 위한 설계 — 이 백서에서는 기업이 HIPAA 적격 애플리케이션을 만드는 AWS 데 사용할 수 있는 방법을 설명합니다.](#)

#### Note

모든 AWS 서비스 사람이 HIPAA 자격을 갖춘 것은 아닙니다. 자세한 내용은 [HIPAA 적격 서비스 참조](#)를 참조하십시오.

- [AWS 규정 준수 리소스AWS](#) — 이 워크북 및 가이드 모음은 해당 산업 및 지역에 적용될 수 있습니다.
- [AWS 고객 규정 준수 가이드](#) — 규정 준수의 관점에서 공동 책임 모델을 이해하십시오. 이 가이드에서는 보안을 유지하기 위한 모범 사례를 AWS 서비스 요약하고 여러 프레임워크 (미국 표준 기술 연구소 (NIST), 결제 카드 산업 보안 표준 위원회 (PCI), 국제 표준화기구 (ISO) 등) 에서 보안 제어에 대한 지침을 매핑합니다.
- 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 내부 AWS보안 상태를 포괄적으로 파악할 수 있습니다. Security Hub는 보안 제어를 사용하여 AWS 리소스를 평가하고 보안 업계 표준 및 모범 사례에 대한 규정 준수를 확인합니다. 지원되는 서비스 및 제어 목록은 [Security Hub 제어 참조](#)를 참조하십시오.
- [Amazon GuardDuty](#) — 환경에 의심스럽고 악의적인 활동이 있는지 AWS 계정모니터링하여 워크로드, 컨테이너 및 데이터에 대한 잠재적 위협을 AWS 서비스 탐지합니다. GuardDuty 특정 규정 준수 프레임워크에서 요구하는 침입 탐지 요구 사항을 충족하여 PCI DSS와 같은 다양한 규정 준수 요구 사항을 해결하는 데 도움이 될 수 있습니다.
- [AWS Audit Manager](#) — 이를 AWS 서비스 통해 AWS 사용량을 지속적으로 감사하여 위협을 관리하고 규정 및 업계 표준을 준수하는 방법을 단순화할 수 있습니다.

## AWS X-Ray의 복원성

AWS 글로벌 인프라는 AWS 리전 및 가용 영역을 중심으로 구축됩니다. AWS 리전에서는 물리적으로 분리되고 격리된 다수의 가용 영역을 제공하며 이러한 가용 영역은 짧은 대기 시간, 높은 처리량 및 높은

은 중복성을 갖춘 네트워크에 연결되어 있습니다. 가용 영역을 사용하면 중단 없이 가용 영역 간에 자동으로 장애 조치가 이루어지는 애플리케이션 및 데이터베이스를 설계하고 운영할 수 있습니다. 가용 영역은 기존의 단일 또는 다중 데이터 센터 인프라보다 가용성, 내결함성, 확장성이 뛰어납니다.

AWS 리전 및 가용 영역에 대한 자세한 내용은 [AWS 글로벌 인프라](#)를 참조하세요.

## AWS X-Ray의 인프라 보안

관리형 서비스인 AWS X-Ray는 AWS 글로벌 네트워크 보안으로 보호됩니다. AWS 보안 서비스와 AWS의 인프라 보호 방법에 대한 자세한 내용은 [AWS 클라우드 보안](#)을 참조하십시오. 인프라 보안에 대한 모범 사례를 사용하여 AWS 환경을 설계하려면 보안 원칙 AWS Well-Architected Framework의 [인프라 보호](#)를 참조하세요.

AWS에서 게시한 API 직접 호출을 사용하여 네트워크를 통해 X-Ray에 액세스합니다. 고객은 다음을 지원해야 합니다.

- 전송 계층 보안(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 주체와 관련된 보안 액세스 키를 사용하여 서명해야 합니다. 또는 [AWS Security Token Service](#)(AWS STS)를 사용하여 임시 보안 자격 증명을 생성하여 요청에 서명할 수 있습니다.

## VPC 엔드포인트와 함께 AWS X-Ray 사용

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

Amazon VPC란 사용자가 정의한 가상 네트워크에서 AWS 리소스를 시작할 때 사용할 수 있는 AWS 서비스입니다. VPC를 사용하여 IP 주소 범위, 서브넷, 라우팅 테이블, 네트워크 게이트웨이 등의 네트워크 설정을 제어할 수 있습니다. VPC를 X-Ray에 연결하려면 [인터페이스 VPC 엔드포인트](#)를 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 무엇입니까](#)를 참조하세요.

인터페이스 VPC 엔드포인트는 프라이빗 IP 주소와 함께 탄력적 네트워크 인터페이스를 사용하여 AWS 서비스 간 프라이빗 통신을 사용할 수 있는 AWS 기술인 AWS PrivateLink에 의해 구동됩니다. 자세한 내용은 Amazon VPC 사용 설명서의 [신규 — AWS PrivateLink AWS 서비스](#) 블로그 게시물 및 [시작하기](#)를 참조하십시오.

선택한 AWS 리전 항목에 X-Ray용 VPC 엔드포인트를 생성할 수 있는지 확인하려면 [지원되는 리전](#)를 참조하십시오.

## 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. AWS X-Ray 서비스 이름을 검색하여 선택합니다: `com.amazonaws.region.xray`.


**Service category**  AWS services  
 Find service by name  
 Your AWS Marketplace services

**Service Name** `com.amazonaws.us-west-2.xray` ⓘ

The screenshot shows a search bar with the text 'Filter by attributes or search by keyword'. Below the search bar is a table with three columns: 'Service Name', 'Owner', and 'Type'. The table contains three rows, with the third row selected (highlighted in blue).

Service Name	Owner	Type
<input type="radio"/> com.amazonaws.us-west-2.transfer.server	amazon	Interface
<input type="radio"/> com.amazonaws.us-west-2.workspaces	amazon	Interface
<input checked="" type="radio"/> com.amazonaws.us-west-2.xray	amazon	Interface

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.

VPC\*   

Subnets  

Availability Zone	Subnet ID
<input checked="" type="checkbox"/> us-west-2a (usw2-az1)	subnet-40d87938
<input type="checkbox"/> us-west-2b (usw2-az2)	subnet-ff4281b5
<input type="checkbox"/> us-west-2c (usw2-az3)	subnet-d14bfb8c
<input type="checkbox"/> us-west-2d (usw2-az4)	subnet-1faf8734

- (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.
- 보안 그룹을 지정하여 엔드포인트 네트워크 인터페이스와 연결합니다.

Security group  [Create a new security group](#) 

Select security groups ▲

1 to 5 of 5

<input type="checkbox"/>	Group ID	Group Name	VPC ID		Description	Owner ID
<input type="checkbox"/>	sg-0683c...	ssh-http	vpc-4f6e3a37	EC2-VPC	launch-wizar...	979300271395
<input type="checkbox"/>	sg-0774...	awseb-e-7xv5...	vpc-4f6e3a37	EC2-VPC	SecurityGrou...	979300271395
<input type="checkbox"/>	sg-0a46...	launch-wizard-1	vpc-4f6e3a37	EC2-VPC	launch-wizar...	979300271395
<input type="checkbox"/>	sg-0d62...	awseb-e-7xv5...	vpc-4f6e3a37	EC2-VPC	Elastic Beans...	979300271395
<input checked="" type="checkbox"/>	sg-d4f14...	default	vpc-4f6e3a37	EC2-VPC	default VPC s...	979300271395

[Close](#)

- (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 제어](#)를 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 생성](#)의 과정을 따르십시오.
4. com.amazonaws.**region**.xray 엔드포인트를 선택한 다음 정책 탭을 선택합니다.
5. 정책 편집을 선택한 다음 변경합니다.

## 지원되는 리전

현재 X-Ray가 VPC 엔드포인트를 지원하는 AWS 리전은 다음과 같습니다.

- 미국 동부(오하이오)
- 미국 동부(버지니아 북부)



- 미국 서부(캘리포니아 북부)
- 미국 서부(오리건)
- 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)

## 에 대한 문서 기록 AWS X-Ray

다음 표에는 에 대한 설명서의 중요한 변경 사항이 설명되어 AWS X-Ray 있습니다. 이 설명서에 대한 업데이트 알림을 받으려면 RSS 피드를 구독하면 됩니다.

최종 설명서 업데이트: 2023년 2월 8일

변경 사항	설명	날짜
<a href="#">추가된 기능</a>	이제 X-Ray는 PutTraceSegments, GetTraceSummaries, 및 BatchGetTraces를 포함한 데이터 이벤트를 AWS CloudTrail 기록합니다. 또한 이제 X-Ray는 GetSamplingStatisticSummaries 관리 이벤트를 CloudTrail 기록합니다. 자세한 내용은 <a href="#">X-Ray API 호출 로깅</a> 을 참조하십시오 AWS CloudTrail.	2024년 3월 7일
<a href="#">추가된 기능</a>	X-Ray는 이제 <a href="#">W3C Trace Context 사양을 준수하는 OpenTelemetry 또는 다른 프레임워크를 통해 생성된 추적 ID</a> 를 지원합니다. 자세한 내용은 <a href="#">X-Ray로 트레이스 데이터 전송</a> 을 참조하십시오.	2023년 10월 25일
<a href="#">추가된 기능</a>	이제 Amazon SNS 활성 추적을 구성하여 요청이 Amazon SNS 토픽을 통과할 때 요청을 추적하고 분석할 수 있습니다. 자세한 내용은 <a href="#">Amazon SNS 및</a> 을 참조하십시오 AWS X-Ray.	2023년 2월 8일

<a href="#"><u>Node.js용 X-Ray SDK 주제 업데이트</u></a>	V3를 사용하는 클라이언트 계측에 대한 세부 정보가 추가되었습니다. AWS SDK for JavaScript 자세한 내용은 <a href="#"><u>Node.js용 X-Ray SDK로 AWS SDK 호출 추적하기</u></a> 를 참조하십시오.	2023년 2월 7일
<a href="#"><u>IAM 관리형 정책 세부 정보 업데이트</u></a>	AWSXRayReadOnlyAccess , AWSXRayFullAccess 및 AWSXrayCrossAccountSharingConfiguration 의 관리형 정책에 계정 간 관찰성을 위한 IAM 권한을 추가했습니다. 자세한 내용은 <a href="#"><u>X-Ray의 IAM 관리형 정책</u></a> 을 참조하십시오.	2023년 2월 7일
<a href="#"><u>추가된 기능</u></a>	AWS X-Ray 이제 계정 간 관찰 기능을 지원하므로 한 계정 내 여러 계정에 걸쳐 있는 애플리케이션을 모니터링하고 문제를 해결할 수 있습니다. AWS 리 전자세한 내용은 <a href="#"><u>교차 계정 추적</u></a> 을 참조하십시오.	2022년 11월 27일
<a href="#"><u>추가된 기능</u></a>	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 <a href="#"><u>이벤트 기반 애플리케이션 추적</u></a> 을 참조하십시오.	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 정책](#)을 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 사용](#)을 참조하십시오.

2022년 11월 11일

## [루비용 AWS 배포판이 추가되었습니다. OpenTelemetry](#)

AWS Distro for OpenTelemetry (ADOT) 는 분산된 트레이스 및 메트릭을 수집할 수 있는 단일 오픈 소스 API, 라이브러리 및 에이전트 세트를 제공합니다. ADOT Ruby를 사용하면 X-Ray 및 기타 추적 백엔드용 Ruby 애플리케이션을 계속할 수 있습니다. 자세한 내용은 Ruby용 [AWS 배포판](#)을 참조하십시오. OpenTelemetry

2022년 2월 7일

## [추가된 기능](#)

이제 CloudWatch 콘솔에서 트레이스를 보고 X-Ray를 구성할 수 있습니다. 자세한 내용은 [X-Ray 콘솔](#) 섹션을 참조하십시오.

2022년 1월 24일

<a href="#">통합 CloudWatch RUM</a>	AWS X-Ray 및 CloudWatch RUM을 사용하면 애플리케이션의 최종 사용자부터 시작하여 다운스트림 AWS 관리 서비스에 이르는 요청 경로를 분석하고 디버깅할 수 있습니다. 자세한 내용은 RUM 및 을 참조하십시오 <a href="#">CloudWatch . AWS X-Ray</a>	2021년 12월 3일
<a href="#">통합 AWS 배포판: OpenTelemetry</a>	AWS 배포판 OpenTelemetry (ADOT) 은 분산된 트레이스 및 메트릭을 수집할 수 있는 단일 오픈 소스 API, 라이브러리 및 에이전트 세트를 제공합니다. ADOT를 사용하면 X-Ray 및 기타 추적 백엔드용 애플리케이션을 계측할 수 있습니다. 자세한 내용은 <a href="#">애플리케이션 계측하기</a> 를 참조하세요.	2021년 9월 23일
<a href="#">추가된 기능</a>	AWS X-Ray 이제 Amazon Virtual Private Cloud와 통합되어 Amazon VPC의 리소스가 공용 인터넷을 거치지 않고도 X-Ray 서비스와 통신할 수 있습니다. 자세한 내용은 <a href="#">VPC 엔드포인트와 함께 AWS X-Ray 사용을 참조하세요.</a>	2021년 5월 20일
<a href="#">추가된 기능</a>	AWS X-Ray 이제 와 AWS CloudFormation 통합되어 X-Ray 리소스를 프로비저닝하고 구성할 수 있습니다. 자세한 내용은 <a href="#">사용하여 X-Ray 리소스 만들기를 참조하십시오</a> CloudFormation.	2021년 5월 6일

<a href="#"><u>추가된 기능</u></a>	AWS X-Ray 이제 EventBridge Amazon과 통합되어 EventBridge 전달된 이벤트를 추적합니다. 이를 통해 사용자는 시스템을 보다 완벽하게 파악할 수 있습니다. 자세한 내용은 <a href="#"><u>Amazon EventBridge 및</u></a> 을 참조하십시오 AWS X-Ray.	2021년 3월 2일
<a href="#"><u>ECR에 대몬(daemon)을 추가했습니다.</u></a>	이제 Amazon ECR에서 대몬(daemon)을 다운로드할 수 있습니다. 자세한 내용은 <a href="#"><u>대몬(daemon) 다운로드하기</u></a> 를 참조하세요.	2021년 3월 1일
<a href="#"><u>추가된 기능</u></a>	AWS X-Ray 이제 Amazon에 대한 인사이트 관련 알림을 지원합니다 EventBridge. 이를 통해 를 사용하여 인사이트에 대한 자동 조치를 취할 수 EventBridge 있습니다. 자세한 내용은 <a href="#"><u>Use X-Ray Insights에서 Insights</u></a> 알림 활성화를 참조하십시오.	2020년 10월 15일
<a href="#"><u>다운로드 가능한 대몬(daemon)이 추가되었습니다.</u></a>	AWS X-Ray Linux ARM64 지원 데몬을 소개합니다. 자세한 내용은 <a href="#"><u>AWS X-Ray daemonbrazil ws</u></a> 를 참조하십시오.	2020년 10월 1일

추가된 기능

AWS X-Ray 이제 Amazon CloudWatch Synthetics와의 적극적인 통합이 지원됩니다. 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 사용한 [CloudWatch 합성 카나리아 디버깅](#)을 참조하십시오.

2020년 9월 24일

추가된 기능

AWS X-Ray 이제 에 대한 추적 워크플로가 지원됩니다. end-to-end 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 계정의 추적 데이터를 지속적으로 분석하여 애플리케이션에서 발생하는 긴급 문제를 식별할 수 있는 통찰력을 소개합니다. Insights는 사고를 기록하고 해결될 때까지 사고 영향을 추적합니다. 자세한 내용은 [X-Ray Insights 사용](#)을 참조하십시오.

2020년 9월 3일

추가된 기능

AWS X-Ray 고객이 기존 Java 기반 애플리케이션을 수정하지 않고도 추적 데이터를 수집할 수 있도록 하는 Java 자동 계측 에이전트를 소개합니다. 이제 구성 변경을 최소화하고 코드를 변경하지 않고도 Java 웹 및 서브렛 기반 응용 프로그램을 추적할 수 있습니다. 자세한 내용은 [Java용 AWS X-Ray 자동 계측 에이전트](#)를 참조하십시오.

2020년 9월 3일

추가된 기능

AWS X-Ray 트레이스 그룹을 쉽게 생성하고 관리할 수 있도록 X-Ray 콘솔에 새 그룹 페이지를 추가했습니다. 자세한 내용은 X-Ray 콘솔의 [그룹 구성](#)을 참조하십시오.

2020년 8월 24일

추가된 기능

AWS X-Ray 이제 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 관리](#)를 참조하십시오.

2020년 8월 24일



기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.