



Guide du développeur

AWS X-Ray



AWS X-Ray: Guide du développeur

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

Les marques commerciales et la présentation commerciale d'Amazon ne peuvent pas être utilisées en relation avec un produit ou un service extérieur à Amazon, d'une manière susceptible d'entraîner une confusion chez les clients, ou d'une manière qui dénigre ou discrédite Amazon. Toutes les autres marques commerciales qui ne sont pas la propriété d'Amazon appartiennent à leurs propriétaires respectifs, qui peuvent ou non être affiliés ou connectés à Amazon, ou sponsorisés par Amazon.

Table of Contents

Qu'est-ce que c'est AWS X-Ray ?	1
Premiers pas	4
Choix d'une interface	6
Utiliser un SDK	7
Utilisez la commande ADOT SDK	8
Utilisez la commande X-Ray SDK	10
Utiliser une console	11
Utiliser la CloudWatch console Amazon	12
Utiliser la console X-Ray	13
Découvrez la console X-Ray	13
Carte de traçage	15
Suivis	22
Expressions de filtrage	32
Traçabilité entre comptes	44
Suivi des applications pilotées par des événements	48
Histogrammes	52
Informations	54
Analyse	63
Groups	70
Echantillonnage	80
Liens profonds sur console	86
Utiliser l'API X-Ray	89
didacticiel	91
Envoi de données	96
Extraction de données	101
Configuration	116
Echantillonnage	123
Documents de segment	127
Concepts	148
Segments	148
Sous-segments	149
Graphique de services	153
Suivis	155
Echantillonnage	156

En-tête de suivi	156
Expressions de filtrage	157
Groups	158
Annotations et métadonnées	159
Erreurs, anomalies et exceptions	159
Sécurité	161
.....	161
Protection des données	161
Gestion des identités et des accès	164
Public ciblé	164
Authentification par des identités	165
Gestion des accès à l'aide de politiques	169
Comment AWS X-Ray fonctionne avec IAM	172
Exemples de politiques basées sur l'identité	180
Résolution des problèmes	194
Journalisation et surveillance	197
Validation de conformité	197
Résilience	199
Sécurité de l'infrastructure	199
Points de terminaison d'un VPC	200
Création d'un point de terminaison VPC pour X-Ray	200
Contrôle de l'accès à votre point de terminaison X-Ray VPC	202
Régions prises en charge	203
Exemple d'application	205
Tutoriel Scorekeep	207
Prérequis	208
Installez l'application Scorekeep à l'aide de CloudFormation	209
Génération de données de suivi	210
Consultez la carte de suivi dans le AWS Management Console	211
Configuration des notifications Amazon SNS	219
Exploration de l'exemple d'application	221
Facultatif : stratégie de moindre privilège	226
Nettoyage	228
Étapes suivantes	229
AWS Clients du SDK	230
Sous-segments personnalisés	230

Annotations et métadonnées	231
Clients HTTP	232
Clients SQL	233
AWS Lambda fonctions	235
Nom aléatoire	236
Nœuds	238
Instrumentation du code de démarrage	240
Instrumentation de scripts	243
Instrumentation des clients Web	244
Threads de travail	248
Démon X-Ray	250
Téléchargement du démon	250
Vérification de la signature de l'archive du démon	252
Exécution du démon	253
Autoriser le daemon à envoyer des données à X-Ray	254
Journaux du démon X-Ray	254
Configuration	255
Variables d'environnement prises en charge	256
Utilisation d'options de ligne de commande	256
Utilisation d'un fichier de configuration	257
Exécutez le démon en local	259
Exécution du daemon X-Ray sous Linux	259
Exécution du daemon X-Ray dans un conteneur Docker	260
Exécution du daemon X-Ray sous Windows	261
Exécution du daemon X-Ray sous OS X	262
Sur Elastic Beanstalk	262
Utilisation de l'intégration Elastic Beanstalk X-Ray pour exécuter le daemon X-Ray	263
Téléchargement et exécution manuels du daemon X-Ray (niveau avancé)	265
Sur Amazon EC2	267
Sur Amazon ECS	268
Utilisation de l'image Docker officielle de l'	268
Création et génération d'une image Docker	269
Configurer les options de ligne de commande dans la console Amazon ECS	272
Intégration avec Services AWS	274
AWS Distro pour OpenTelemetry	276
AWS Distro pour OpenTelemetry	276

API Gateway	277
App Mesh	279
App Runner	282
AWS AppSync	282
CloudTrail	282
Événements relatifs à la gestion des rayons X dans CloudTrail	284
Événements liés aux données X-Ray dans CloudTrail	285
Exemples d'événements X-Ray	287
CloudWatch	289
CloudWatch RHUM	290
CloudWatch Synthetics	291
AWS Config	300
Création d'un déclencheur de fonction Lambda	301
Création d'une AWS Config règle personnalisée pour la radiographie	302
Exemples de résultat	303
Notifications Amazon SNS	304
Amazon EC2	304
Elastic Beanstalk	304
Elastic Load Balancing	305
EventBridge	305
Affichage de la source et des cibles sur la carte du service X-Ray	306
Propagation du contexte de suivi vers les cibles des événements	306
Lambda	312
Amazon SNS	314
Configurer le suivi actif Amazon SNS	315
Afficher les traces des éditeurs et des abonnés Amazon SNS dans la console X-Ray	316
Step Functions	318
Amazon SQS	319
Envoi de l'en-tête de suivi HTTP	321
Récupération de l'en-tête de suivi et récupération du contexte de suivi	321
Amazon S3	322
Configuration des notifications d'événements Amazon S3	323
Instrumentation de votre application	325
Instrumentation de votre application avec le AWS Distro pour OpenTelemetry	325
Instrumentation de votre application avec AWS X-Ray SDKs	327
Choisir entre The AWS Distro for OpenTelemetry et X-Ray SDKs	328

Recherche de transactions	330
OpenTelemetry Point de terminaison du protocole (OTLP)	331
Utilisation de Go	332
AWS Distro pour Go OpenTelemetry	332
Kit de développement X-Ray pour Go	332
Prérequis	334
Documentation de référence	334
Configuration	335
Demandes entrantes	341
AWS Clients du SDK	344
Appels HTTP sortants	346
Requêtes SQL	347
Sous-segments personnalisés	347
Annotations et métadonnées	348
Utilisation de Java	351
AWS Distro pour Java OpenTelemetry	351
Kit de développement X-Ray pour Java	351
Sous-modules	353
Prérequis	354
Gestion des dépendances	355
Agent d'auto-instrumentation	357
Configuration	370
Demandes entrantes	382
AWS Clients du SDK	386
Appels HTTP sortants	389
Requêtes SQL	391
Sous-segments personnalisés	393
Annotations et métadonnées	396
Surveillance	401
Multithreading	405
AOP avec Spring	406
Utilisation de Node.js	411
AWS Distro pour OpenTelemetry JavaScript	411
Kit de développement X-Ray pour Node.js	412
Prérequis	414
Gestion des dépendances	414

Exemples Node.js	415
Configuration	415
Demandes entrantes	421
AWS Clients du SDK	425
Appels HTTP sortants	429
Requêtes SQL	430
Sous-segments personnalisés	432
Annotations et métadonnées	434
Travail avec Python	439
AWS Distro pour Python OpenTelemetry	439
X-Ray SDK pour Python	439
Prérequis	442
Gestion des dépendances	443
Configuration	443
Demandes entrantes	450
Correction des bibliothèques	456
AWS Clients du SDK	459
Appels HTTP sortants	460
Sous-segments personnalisés	462
Annotations et métadonnées	463
Instrumentation des applications sans serveur	467
Utilisation de .NET	474
AWS Distro pour .NET OpenTelemetry	474
Kit de développement X-Ray pour .NET	474
Prérequis	477
Ajout du SDK X-Ray pour .NET à votre application	477
Gestion des dépendances	477
Configuration	479
Demandes entrantes	486
AWS Clients du SDK	490
Appels HTTP sortants	492
Requêtes SQL	494
Sous-segments personnalisés	497
Annotations et métadonnées	498
Utilisation de Ruby	502
AWS Distro pour Ruby OpenTelemetry	502

Kit de développement X-Ray pour Ruby	502
Prérequis	504
Configuration	504
Demandes entrantes	511
Correction des bibliothèques	515
AWS Clients du SDK	516
Sous-segments personnalisés	517
Annotations et métadonnées	518
Migration de l'instrumentation X-Ray vers l' OpenTelemetry instrumentation	522
Compréhension OpenTelemetry	522
OpenTelemetry soutien dans AWS	523
Comprendre les OpenTelemetry concepts relatifs à la migration	524
Comparaison des fonctionnalités	525
Configuration et configuration du suivi	526
Détecter les ressources dans votre environnement	527
Gestion des stratégies d'échantillonnage	528
Gestion du contexte de suivi	529
Propagation du contexte de trace	529
Utilisation de l'instrumentation de bibliothèque	530
Exporter des traces	530
Traitement et transmission des traces	531
Traitement de l'OpenTelemetry envergure (concept spécifique)	532
OpenTelemetry Bagages (concept spécifique)	532
Présentation de la migration	532
Recommandations pour les applications nouvelles et existantes	533
Modification de la configuration du suivi	533
Modifications apportées aux instruments de bibliothèque	534
Modifications apportées à l'instrumentation de l'environnement Lambda	534
Création manuelle de données de trace	535
Migration de X-Ray Daemon vers un AWS CloudWatch agent ou un collecteur	
OpenTelemetry	535
Migration sur Amazon EC2 ou sur des serveurs sur site	536
Migration sur Amazon ECS	540
Migration sur Elastic Beanstalk	544
Migration vers Java OpenTelemetry	545
Solution d'instrumentation automatique à code zéro	546

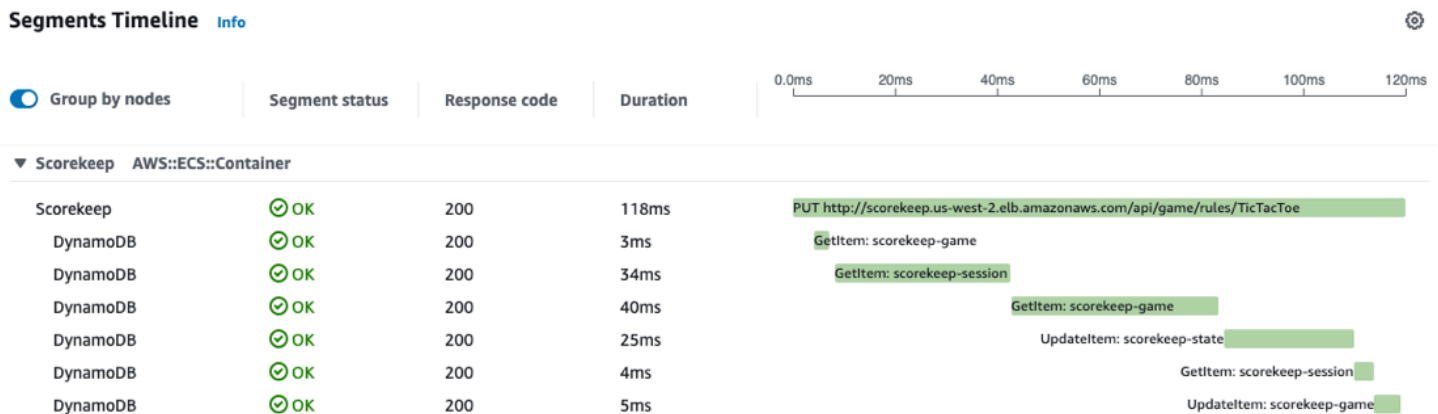
Solutions d'instrumentation manuelle avec le SDK	546
Suivi des demandes entrantes (instrumentation Spring Framework)	549
AWS Instrumentation du SDK v2	550
Instrumentation des appels HTTP sortants	552
Support d'instrumentation pour d'autres bibliothèques	553
Création manuelle de données de trace	553
Instrumentation Lambda	556
Migrer vers OpenTelemetry Go	562
Instrumentation manuelle avec le SDK	562
Suivi des demandes entrantes (instrumentation du gestionnaire HTTP)	564
AWS SDK pour l'instrumentation du SDK Go v2	565
Instrumentation des appels HTTP sortants	566
Support d'instrumentation pour d'autres bibliothèques	568
Création manuelle de données de trace	568
Instrumentation manuelle Lambda	570
Migrer vers OpenTelemetry Node.js	576
Solutions d'instrumentation automatique à code zéro	577
Solutions d'instrumentation manuelle	577
Suivi des demandes entrantes	581
AWS Instrumentation du SDK JavaScript V3	565
Instrumentation des appels HTTP sortants	584
Support d'instrumentation pour d'autres bibliothèques	585
Création manuelle de données de trace	568
Instrumentation Lambda	570
Migrer vers OpenTelemetry .NET	588
Solutions d'instrumentation automatique à code zéro	589
Solutions d'instrumentation manuelle avec le SDK	590
Création manuelle de données de trace	593
Suivi des demandes entrantes (ASP.NET et instrumentation de base ASP.NET)	596
AWS Instrumentation du SDK	597
Instrumentation des appels HTTP sortants	598
Support d'instrumentation pour d'autres bibliothèques	599
Instrumentation Lambda	570
Migrer vers OpenTelemetry Python	603
Solutions d'instrumentation automatique à code zéro	604
Instrumentez manuellement vos applications	604

Initialisation de la configuration de suivi	605
Suivi des demandes entrantes	608
AWS Instrumentation du SDK	609
Instrumentation des appels HTTP sortants par le biais de requêtes	611
Support d'instrumentation pour d'autres bibliothèques	612
Création manuelle de données de trace	612
Instrumentation Lambda	614
Migrer vers OpenTelemetry Ruby	615
Instrumentez manuellement vos solutions avec le SDK	616
Suivi des demandes entrantes (instrumentation Rails)	619
AWS Instrumentation du SDK	620
Instrumentation des appels HTTP sortants	620
Support d'instrumentation pour d'autres bibliothèques	621
Création manuelle de données de trace	622
Instrumentation manuelle Lambda	624
Création de ressources X-Ray avec CloudFormation	628
X-Ray et AWS CloudFormation modèles	628
En savoir plus sur AWS CloudFormation	628
Identification	629
Restrictions liées aux étiquettes	630
Gestion des tags dans la console	631
Ajouter des tags à un nouveau groupe (console)	631
Ajouter des balises à une nouvelle règle d'échantillonnage (console)	632
Modifier ou supprimer les balises d'un groupe (console)	632
Modifier ou supprimer des balises pour une règle d'échantillonnage (console)	633
Gestion des balises dans AWS CLI	633
Ajouter des balises à un nouveau groupe X-Ray ou à une nouvelle règle d'échantillonnage (CLI)	634
Ajouter des balises à une ressource existante (CLI)	636
Répertorier les balises d'une ressource (CLI)	636
Supprimer des balises sur une ressource (CLI)	637
Contrôlez l'accès aux ressources X-Ray en fonction des balises	637
Résolution des problèmes	639
Carte de traçage X-Ray et pages de détails des traces	639
Je ne vois pas tous mes CloudWatch journaux	639
Je ne vois pas toutes mes alarmes sur la carte X-Ray Trace	640

Je ne vois pas certaines AWS ressources sur la carte de suivi	640
Il y a trop de nœuds sur la carte de trace	641
Kit de développement X-Ray pour Java	641
Kit de développement X-Ray pour Node.js	641
Le démon X-Ray	642
Historique du document	643
.....	dcliii

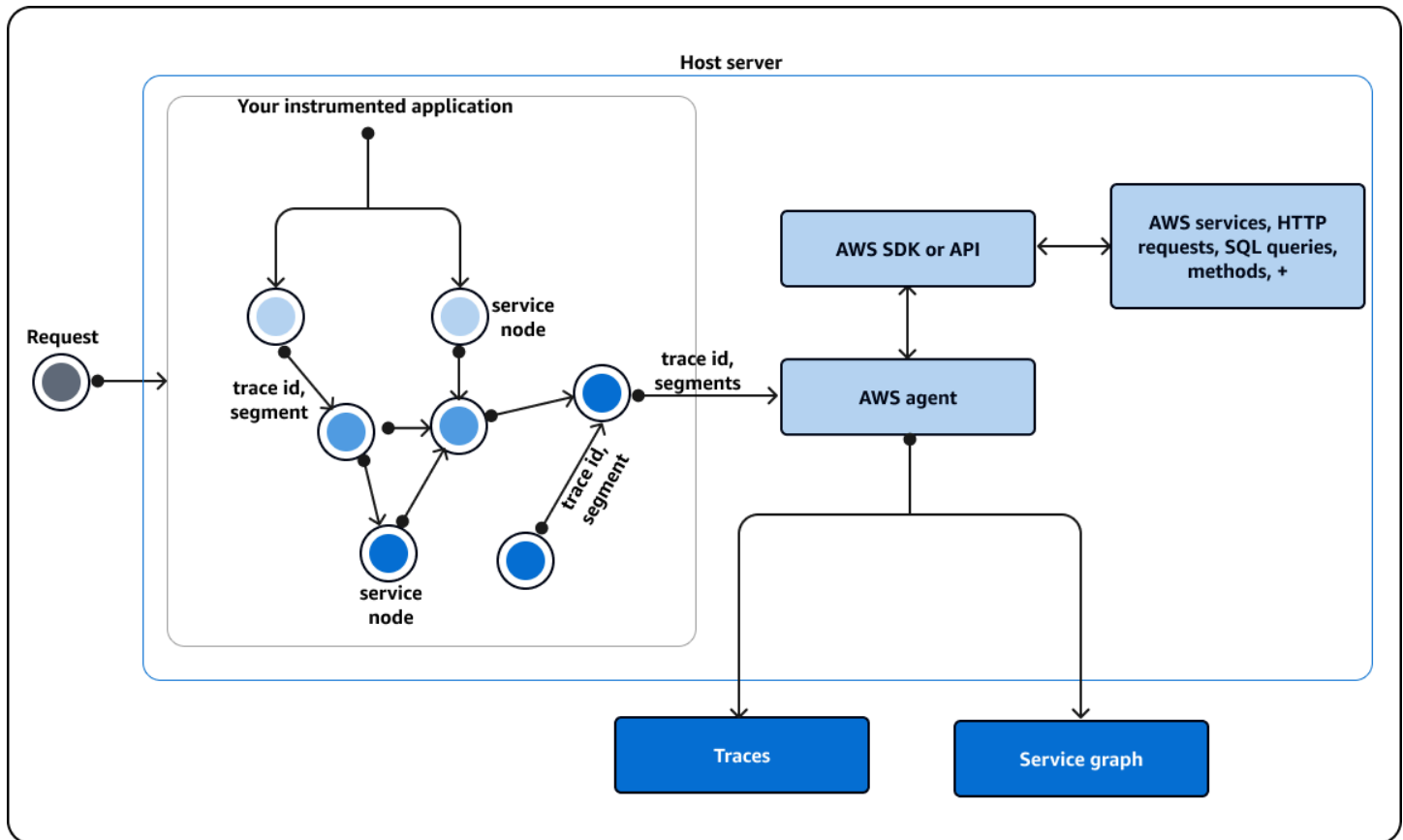
Qu'est-ce que c'est AWS X-Ray ?

AWS X-Ray est un service qui collecte des données sur les demandes traitées par votre application et fournit des outils que vous pouvez utiliser pour afficher, filtrer et obtenir des informations sur ces données afin d'identifier les problèmes et les opportunités d'optimisation. Pour toute demande retracée envoyée à votre application, vous pouvez consulter des informations détaillées non seulement sur la demande et la réponse, mais également sur les appels que votre application fait aux AWS ressources en aval, aux microservices, aux bases de données et au Web APIs.



AWS X-Ray reçoit des traces de votre application, en plus des utilisations de Services AWS votre application qui sont déjà intégrées à X-Ray. L'instrumentation de votre application implique l'envoi de données de suivi pour les demandes entrantes et sortantes et d'autres événements au sein de votre application, ainsi que des métadonnées relatives à chaque demande. De nombreux scénarios d'instrumentation exigent uniquement des modifications de configuration. Par exemple, vous pouvez instrumenter toutes les requêtes HTTP entrantes et les appels en Services AWS aval effectués par votre application Java. Il existe plusieurs SDKs agents et outils qui peuvent être utilisés pour instrumenter votre application pour le traçage par rayons X. Voir [Instrumentation de votre application](#) pour plus d'informations.

Services AWS qui sont [intégrés à X-Ray](#) peuvent ajouter des en-têtes de suivi aux demandes entrantes, envoyer des données de suivi à X-Ray ou exécuter le daemon X-Ray. AWS Lambda Vous pouvez, par exemple, envoyer des données de suivi concernant les requêtes adressées à vos fonctions Lambda et exécuter le démon X-Ray sur les utilisateurs afin de simplifier l'utilisation du SDK X-Ray.



Au lieu d'envoyer les données de suivi directement à X-Ray, chaque SDK client envoie des documents de segment JSON à un processus démon qui écoute le trafic UDP. Le [daemon X-Ray](#) met en mémoire tampon les segments d'une file d'attente et les télécharge dans X-Ray par lots. Le daemon est disponible pour Linux, Windows et macOS, et est inclus sur les AWS Elastic Beanstalk AWS Lambda plateformes.

X-Ray utilise les données de suivi provenant des AWS ressources qui alimentent vos applications cloud pour générer une carte de suivi détaillée. La carte de suivi indique le client, votre service frontal et les services principaux que votre service frontal appelle pour traiter les demandes et conserver les données. Utilisez la carte de suivi pour identifier les goulots d'étranglement, les pics de latence et les autres problèmes à résoudre afin d'améliorer les performances de vos applications.



Commencer à utiliser X-Ray

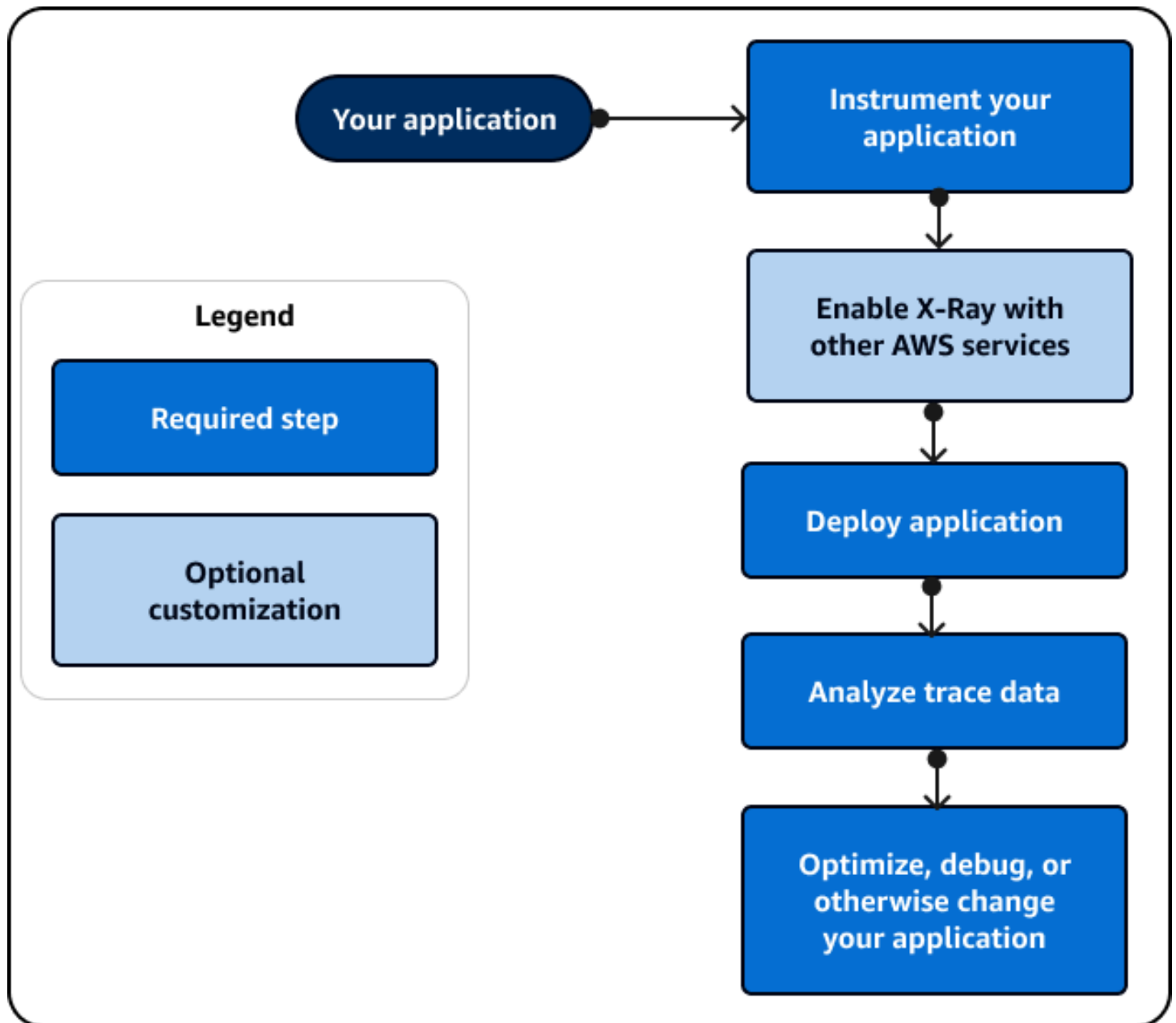
Pour utiliser X-Ray, procédez comme suit :

1. Instrumentez votre application, ce qui permet à X-Ray de suivre la manière dont votre application traite une demande.
 - Utilisez le X-Ray SDKs, le X-Ray APIs, ADOT ou CloudWatch Application Signals pour envoyer des données de suivi à X-Ray. Pour plus d'informations sur l'interface à utiliser, consultez [Choix d'une interface](#).

Pour plus d'informations sur l'instrumentation, voir [Instrumentation de votre application pour AWS X-Ray](#).

2. (Facultatif) Configurez X-Ray pour Services AWS qu'il fonctionne avec d'autres appareils intégrés à X-Ray. Vous pouvez échantillonner des traces et ajouter des en-têtes aux demandes entrantes, exécuter un agent ou un collecteur et envoyer automatiquement les données de trace à X-Ray. Pour de plus amples informations, veuillez consulter [Intégration AWS X-Ray avec d'autres Services AWS](#).
3. Déployez votre application instrumentée. Au fur et à mesure que votre application reçoit des demandes, le SDK X-Ray enregistre les données de trace, de segment et de sous-segment. Au cours de cette étape, vous devrez peut-être également configurer une politique IAM et déployer un agent ou un collecteur.
 - Pour des exemples de scripts permettant de déployer une application à l'aide du SDK AWS Distro for OpenTelemetry (ADOT) et de l' CloudWatch agent sur différentes plateformes, consultez la section Scripts de démonstration [des signaux d'application](#).
 - Pour un exemple de script permettant de déployer une application à l'aide du SDK X-Ray et du daemon X-Ray, consultez. [AWS X-Ray exemple d'application](#)
4. (Facultatif) Ouvrez une console pour afficher et analyser les données. Vous pouvez consulter une représentation graphique d'une carte de trace, d'une carte de service, etc. pour inspecter le fonctionnement de votre application. Utilisez les informations graphiques de la console pour optimiser, déboguer et comprendre votre application. Pour plus d'informations sur le choix d'une console, consultez [Utiliser une console](#).

Le schéma suivant montre comment commencer à utiliser X-Ray :



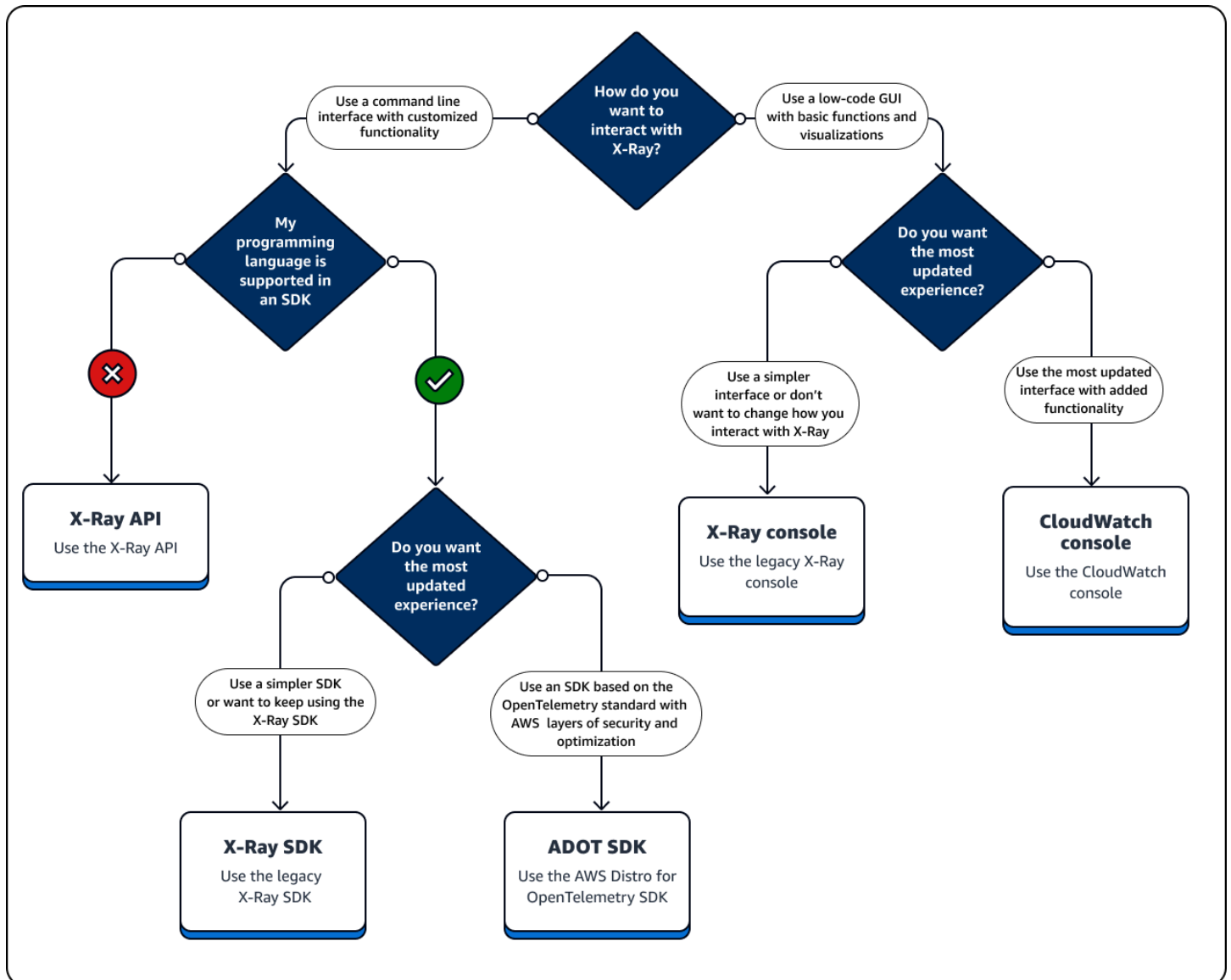
Pour obtenir un exemple des données et des cartes disponibles dans la console, lancez un [exemple d'application](#) déjà instrumenté pour générer des données de suivi. En quelques minutes, vous pouvez générer du trafic, envoyer des segments à X-Ray et consulter une trace et une carte des services.

Choix d'une interface

AWS X-Ray peut fournir des informations sur le fonctionnement de votre application et sur la manière dont elle interagit avec d'autres services et ressources. Une fois que vous avez instrumenté ou configuré votre application, X-Ray collecte des données de suivi lorsque votre application répond aux demandes. Vous pouvez analyser ces données de suivi pour identifier les problèmes de performances, résoudre les erreurs et optimiser vos ressources. Ce guide explique comment interagir avec X-Ray en respectant les consignes suivantes :

- Utilisez-en un AWS Management Console si vous souhaitez démarrer rapidement ou si vous pouvez utiliser des visualisations prédéfinies pour effectuer des tâches de base.
 - Choisissez la CloudWatch console Amazon pour bénéficier de l'expérience utilisateur la plus récente contenant toutes les fonctionnalités de la console X-Ray.
 - Utilisez la console X-Ray si vous souhaitez une interface plus simple ou si vous ne voulez pas changer la façon dont vous interagissez avec X-Ray.
- Utilisez un SDK si vous avez besoin de fonctionnalités de suivi, de surveillance ou de journalisation plus personnalisées que celles qu'un logiciel AWS Management Console peut fournir.
 - Cliquez sur l'onglet ADOT SDK si vous voulez un SDK indépendant du fournisseur basé sur l'open source OpenTelemetry SDK avec couches supplémentaires de AWS sécurité et d'optimisation.
 - Choisissez le SDK X-Ray si vous souhaitez un SDK plus simple ou si vous ne souhaitez pas mettre à jour le code de votre application.
- Utilisez les opérations de l'API X-Ray si un SDK ne prend pas en charge le langage de programmation de votre application.

Le schéma suivant vous aide à choisir le mode d'interaction avec X-Ray :



Découvrez les types d'interfaces

- [Utiliser un SDK](#)
- [Utiliser une console](#)
- [Utiliser l'API X-Ray](#)

Utiliser un SDK

Utilisez un SDK si vous souhaitez utiliser une interface de ligne de commande ou si vous avez besoin de fonctionnalités de suivi, de surveillance ou de journalisation plus personnalisées que celles disponibles dans un AWS Management Console. Vous pouvez également utiliser un AWS SDK pour

développer des programmes utilisant le X-Ray APIs. Vous pouvez utiliser le SDK AWS Distro for OpenTelemetry (ADOT) ou le SDK X-Ray.

Si vous utilisez un SDK, vous pouvez ajouter des personnalisations à votre flux de travail à la fois lorsque vous instrumentez votre application et lorsque vous configurez votre collecteur ou votre agent. Vous pouvez utiliser un SDK pour effectuer les tâches suivantes que vous ne pouvez pas effectuer à l'aide d'un AWS Management Console :

- Publiez des métriques personnalisées : échantillonnez des métriques à des résolutions élevées (jusqu'à 1 seconde), utilisez plusieurs dimensions pour ajouter des informations sur une métrique et agréger des points de données dans un ensemble de statistiques.
- Personnalisez votre collecteur : personnalisez la configuration de n'importe quelle partie d'un collecteur, y compris le récepteur, le processeur, l'exportateur et le connecteur.
- Personnalisez votre instrumentation : personnalisez les segments et les sous-segments, ajoutez des paires clé-valeur personnalisées en tant qu'attributs et créez des métriques personnalisées.
- Créez et mettez à jour des règles d'échantillonnage par programmation.

Utilisez la commande ADOT SDK si vous souhaitez bénéficier de la flexibilité d'utiliser une solution standardisée OpenTelemetry SDK avec couches supplémentaires de AWS sécurité et d'optimisation. The AWS Distro pour OpenTelemetry (ADOT) Le SDK est un package indépendant du fournisseur qui permet l'intégration avec les backends d'autres fournisseurs et non fournisseurs de AWS services sans avoir à réinstrumenter votre code.

Utilisez le SDK X-Ray si vous utilisez déjà le SDK X-Ray, si vous l'intégrez uniquement aux AWS backends et si vous ne souhaitez pas modifier la façon dont vous interagissez avec X-Ray ou le code de votre application.

Pour plus d'informations sur chaque fonctionnalité, consultez [Choisir entre The AWS Distro for OpenTelemetry et X-Ray SDKs](#).

Utilisez la commande ADOT SDK

Le ADOT Le SDK est un ensemble de bibliothèques APIs, de bibliothèques et d'agents open source qui envoient des données aux services principaux. ADOT est pris en charge par plusieurs backends et agents AWS, s'intègre à ceux-ci et fournit un grand nombre de bibliothèques open source gérées par OpenTelemetry communauté. Utilisez la commande ADOT SDK pour instrumenter votre application et collecter des journaux, des métadonnées, des métriques et des traces. Vous

pouvez également utiliser ADOT pour surveiller les services et définir une alarme en fonction de vos indicateurs dans CloudWatch.

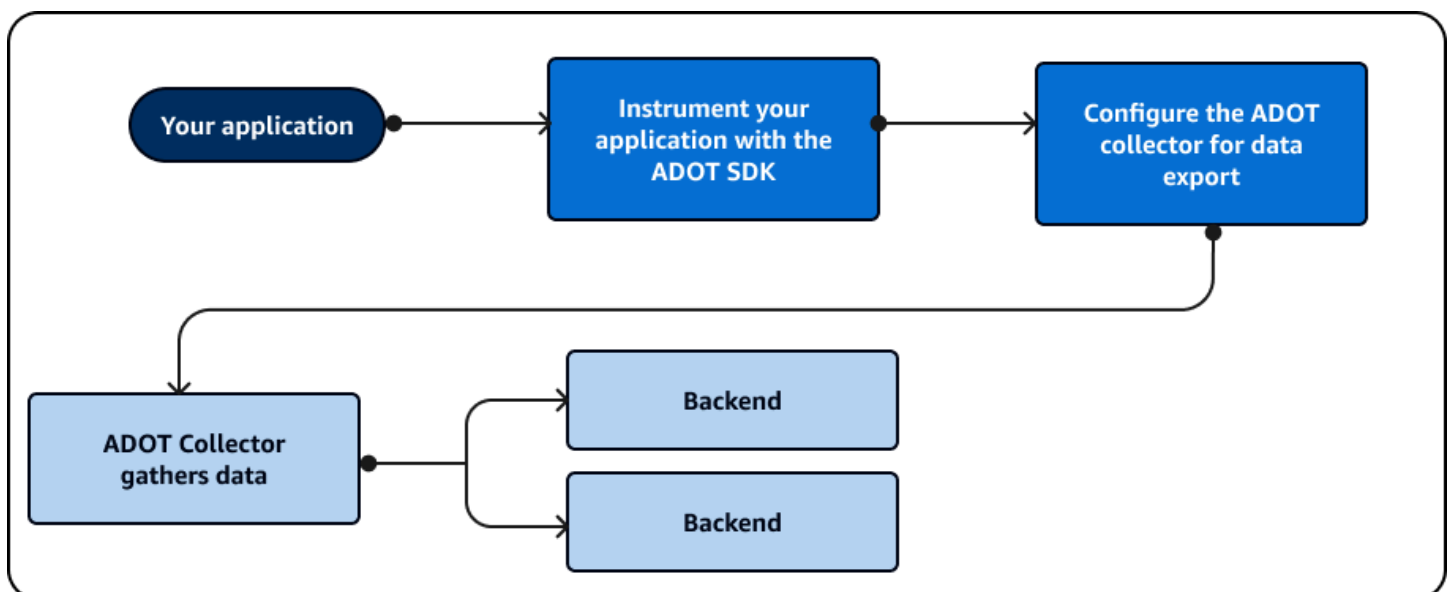
Si vous utilisez le ADOT SDK, vous disposez des options suivantes, en combinaison avec un agent :

- Utilisez la commande ADOT SDK avec [CloudWatch agent](#) : recommandé.
- Utilisez la commande ADOT SDK avec [ADOT Collector](#) : recommandé si vous souhaitez utiliser un logiciel indépendant du fournisseur avec AWS plusieurs niveaux de sécurité et d'optimisation.

Pour utiliser le plugin ADOT SDK, procédez comme suit :

- Instrumentez votre application à l'aide du ADOT SDK. Pour plus d'informations, consultez la documentation de votre langage de programmation dans la [documentation technique ADOT](#).
- Configurez un ADOT collecteur pour lui indiquer où envoyer les données qu'il collecte.

Après le ADOT le collecteur reçoit vos données, il les envoie au backend que vous spécifiez dans ADOT configuration. ADOT peut envoyer des données à plusieurs backends, y compris à des fournisseurs extérieurs AWS, comme le montre le schéma suivant :



AWS mises à jour régulières ADOT pour ajouter des fonctionnalités et s'aligner sur le [OpenTelemetry](#) cadre. Mises à jour et futurs plans de développement ADOT font partie d'une [feuille de route](#) accessible au public. ADOT prend en charge plusieurs langages de programmation, dont les suivants :

- Go

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

Si vous utilisez Python, ADOT peut automatiquement instrumenter votre application. Pour commencer à utiliser ADOT, voir [Présentation](#) et prise [en main de la AWS distribution pour OpenTelemetry Collector](#).

Utilisez la commande X-Ray SDK

Le SDK X-Ray est un ensemble de AWS APIs bibliothèques qui envoient des données aux services AWS principaux. Utilisez le SDK X-Ray pour instrumenter votre application et collecter des données de suivi. Vous ne pouvez pas utiliser le SDK X-Ray pour collecter des données de log ou de métrique.

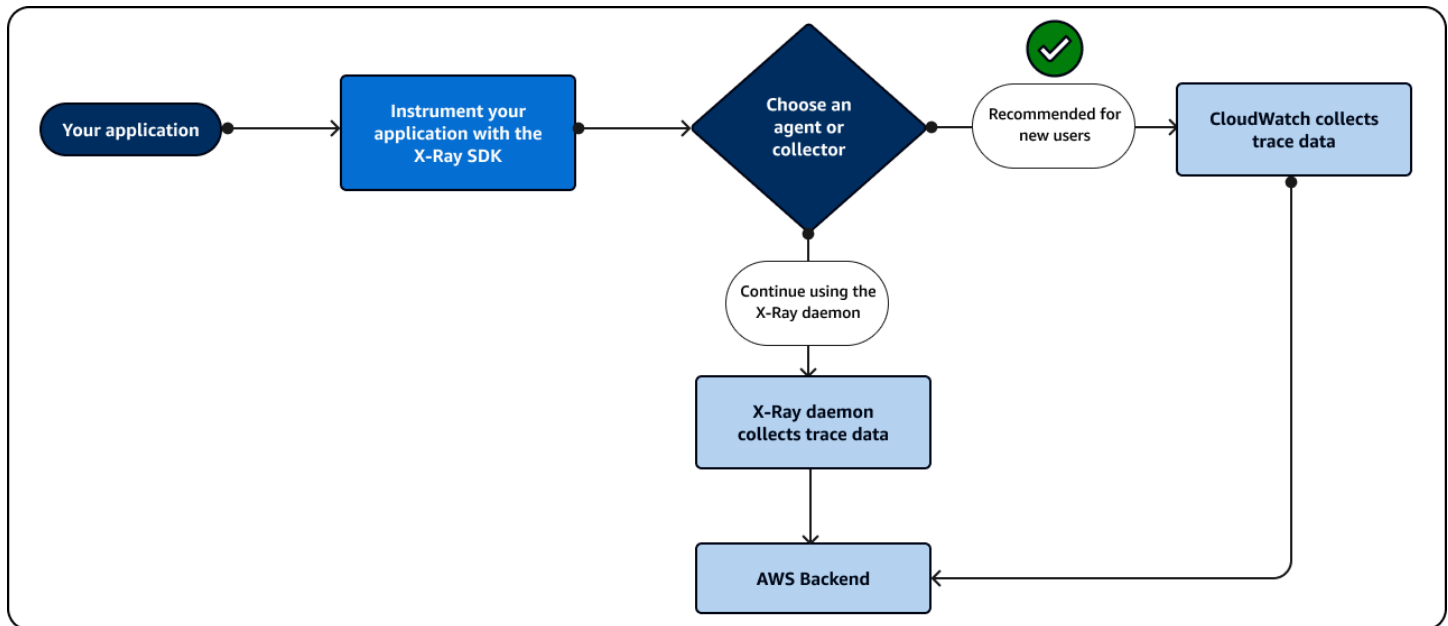
Si vous utilisez le SDK X-Ray, vous disposez des options suivantes, en combinaison avec un agent :

- Utilisez le SDK X-Ray avec le [AWS X-Ray démon](#) — Utilisez-le si vous ne souhaitez pas mettre à jour le code de votre application.
- Utiliser le SDK X-Ray avec l' CloudWatch agent — (recommandé) L' CloudWatch agent est compatible avec le X-Ray SDK.

Pour utiliser le SDK X-Ray, procédez comme suit :

- Instrumentez votre application à l'aide du SDK X-Ray.
- Configurez un collecteur pour lui indiquer où envoyer les données qu'il collecte. Vous pouvez utiliser l' CloudWatch agent ou le daemon X-Ray pour collecter vos informations de suivi.

Une fois que le collecteur ou l'agent a reçu vos données, il les envoie à un AWS backend que vous spécifiez dans la configuration de l'agent. Le SDK X-Ray peut uniquement envoyer des données à un AWS backend, comme le montre le schéma suivant :



Si vous utilisez Java, vous pouvez utiliser le SDK X-Ray pour instrumenter automatiquement votre application. Pour commencer à utiliser le SDK X-Ray, consultez les bibliothèques associées aux langages de programmation suivants :

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

Utiliser une console

Utilisez une console si vous souhaitez une interface utilisateur graphique (GUI) nécessitant un minimum de codage. Les utilisateurs novices de X-Ray peuvent se lancer rapidement à l'aide de visualisations prédéfinies et à effectuer des tâches de base. Vous pouvez effectuer les opérations suivantes directement depuis la console :

- Activez X-Ray.
- Consultez des résumés de haut niveau des performances de votre application.
- Vérifiez l'état de santé de vos applications.

- Identifiez les erreurs de haut niveau.
- Consultez les résumés des traces de base.

Vous pouvez utiliser la CloudWatch console Amazon <https://console.aws.amazon.com/cloudwatch/> ou la console X-Ray de votre <https://console.aws.amazon.com/xray/domicile> pour interagir avec X-Ray.

Utiliser la CloudWatch console Amazon

La CloudWatch console inclut une nouvelle fonctionnalité X-Ray qui a été repensée par rapport à la console X-Ray pour en faciliter l'utilisation. Si vous utilisez la CloudWatch console, vous pouvez consulter les CloudWatch journaux et les métriques ainsi que les données de suivi de X-Ray. Utilisez la CloudWatch console pour afficher et analyser les données, notamment les suivantes :

- **X-Ray Traces** : visualisez, analysez et filtrez les traces associées à votre application lorsqu'elle répond à une demande. Utilisez ces traces pour détecter les latences élevées, les erreurs de débogage et optimiser le flux de travail de votre application. Consultez une carte de trace et une carte des services pour obtenir des représentations visuelles du flux de travail de votre application.
- **Journaux** : affichez, analysez et filtrez les journaux produits par votre application. Utilisez les journaux pour résoudre les erreurs et configurez la surveillance en fonction de valeurs de journal spécifiques.
- **Métriques** : mesurez et surveillez les performances de vos applications à l'aide de métriques émises par vos ressources ou créez vos propres métriques. Affichez ces indicateurs sous forme de graphiques et de tableaux.
- **Surveillance des réseaux et de l'infrastructure** : surveillez les principaux réseaux pour détecter les pannes, ainsi que l'état et les performances de votre infrastructure, y compris les applications conteneurisées, les autres AWS services et les clients.
- Toutes les fonctionnalités de la console X-Ray sont répertoriées dans la section Utiliser la console X-Ray suivante.

Pour plus d'informations sur la CloudWatch console, consultez [Getting started with Amazon CloudWatch](#).

Connectez-vous à la CloudWatch console Amazon à l'adresse <https://console.aws.amazon.com/cloudwatch/>.

Utiliser la console X-Ray

La console X-Ray propose un suivi distribué pour les demandes d'applications. Utilisez la console X-Ray si vous souhaitez une expérience de console plus simple ou si vous ne souhaitez pas mettre à jour le code de votre application. AWS ne développe plus la console X-Ray. La console X-Ray contient les fonctionnalités suivantes pour les applications instrumentées :

- [Informations](#) : détectez automatiquement les anomalies dans les performances de votre application et identifiez les causes sous-jacentes. Les informations sont incluses dans la CloudWatch console sous Insights. Pour plus d'informations, consultez la section Utiliser X-Ray Insights dans [Utiliser la console X-Ray](#).
- Carte des services : affichez une structure graphique de votre application et de ses connexions avec les clients, les ressources, les services et les dépendances.
- Traces — Consultez un aperçu des traces générées par votre application lorsqu'elle répond à une demande. Utilisez les données de suivi pour comprendre les performances de votre application par rapport aux indicateurs de base, notamment HTTP réponse et temps de réponse.
- Analyses — Interprétez, explorez et analysez les données de suivi à l'aide de graphiques pour la distribution des temps de réponse.
- Configuration — Créez des traces personnalisées pour modifier les configurations par défaut pour les éléments suivants :
 - Échantillonnage : créez une règle qui définit la fréquence à laquelle vous devez échantillonner votre application pour obtenir des informations de suivi. Pour plus d'informations, voir Configurer les règles d'échantillonnage dans [Utiliser la console X-Ray](#).
 - [Chiffrement](#) — Chiffrez les données au repos à l'aide d'une clé que vous pouvez auditer ou désactiver à l'aide AWS Key Management Service d'une clé.
 - Groupes : utilisez une expression de filtre pour définir un groupe de traces présentant une caractéristique commune telle que le nom d'une URL ou un temps de réponse. Pour plus d'informations, consultez la section [Configuration des groupes](#).

Connectez-vous à la console X-Ray à la <https://console.aws.amazon.com/xray/maison>.

Découvrez la console X-Ray

Utilisez la console X-Ray pour afficher une carte des services et des traces associées aux demandes traitées par vos applications, et pour configurer les groupes et les règles d'échantillonnage qui affectent la manière dont les traces sont envoyées à X-Ray.

Note

La carte et CloudWatch ServiceLens la carte X-Ray Service ont été combinées dans la carte de trace X-Ray dans la CloudWatch console Amazon. Ouvrez la [CloudWatchconsole](#) et choisissez Trace Map sous X-Ray Traces dans le volet de navigation de gauche. CloudWatch inclut désormais [Application Signals](#), qui permet de découvrir et de surveiller les services de votre application, vos clients, les canaries Synthetics et les dépendances des services. Utilisez Application Signals pour consulter une liste ou une carte visuelle de vos services, consulter les indicateurs de santé en fonction de vos objectifs de niveau de service (SLOs) et effectuer une analyse descendante pour voir les traces de X-Ray corrélées afin de résoudre les problèmes de manière plus détaillée.

La page principale de la console X-Ray est la carte de trace, qui est une représentation visuelle du graphe de service JSON généré par X-Ray à partir des données de suivi générées par vos applications. La cartographie se compose de nœuds de service pour chaque application de votre compte qui sert des demandes, de nœuds clients en amont qui représentent les origines des demandes et de nœuds de service en aval qui représentent les services web et les ressources utilisées par une application lors du traitement d'une demande. Des pages supplémentaires permettent d'afficher les traces et les détails des traces, ainsi que de configurer les groupes et les règles d'échantillonnage.

Consultez l'expérience de X-Ray sur console et comparez-la avec la CloudWatch console dans les sections suivantes.

Découvrez le X-Ray et les CloudWatch consoles

- [Utilisation de la carte de traçage X-Ray](#)
- [Afficher les traces et les détails des traces](#)
- [Utilisation d'expressions de filtre](#)
- [Traçabilité entre comptes](#)
- [Suivi des applications pilotées par des événements](#)
- [Utilisation d'histogrammes de latence](#)
- [Utilisation des informations issues de X-Ray](#)
- [Interaction avec la console Analytics](#)
- [Configuration des groupes](#)

- [Configuration de règles d'échantillonnage](#)
- [Liens profonds sur console](#)

Utilisation de la carte de traçage X-Ray

Consultez la carte de suivi X-Ray pour identifier les services présentant des erreurs, les connexions présentant une latence élevée ou les traces de demandes infructueuses.

Note

CloudWatch inclut désormais [Application Signals](#), qui permet de découvrir et de surveiller les services, les clients, les canaris synthétiques et les dépendances des services de votre application. Utilisez Application Signals pour consulter une liste ou une carte visuelle de vos services, consulter les indicateurs de santé en fonction de vos objectifs de niveau de service (SLOs) et effectuer une analyse descendante pour voir les traces de X-Ray corrélées afin de résoudre les problèmes de manière plus détaillée.

La carte et CloudWatch ServiceLens la carte du service X-Ray sont combinées dans la carte de suivi X-Ray dans la CloudWatch console Amazon. Ouvrez la [CloudWatch console](#) et choisissez Trace Map sous X-Ray Traces dans le volet de navigation de gauche.

Affichage de la carte de suivi

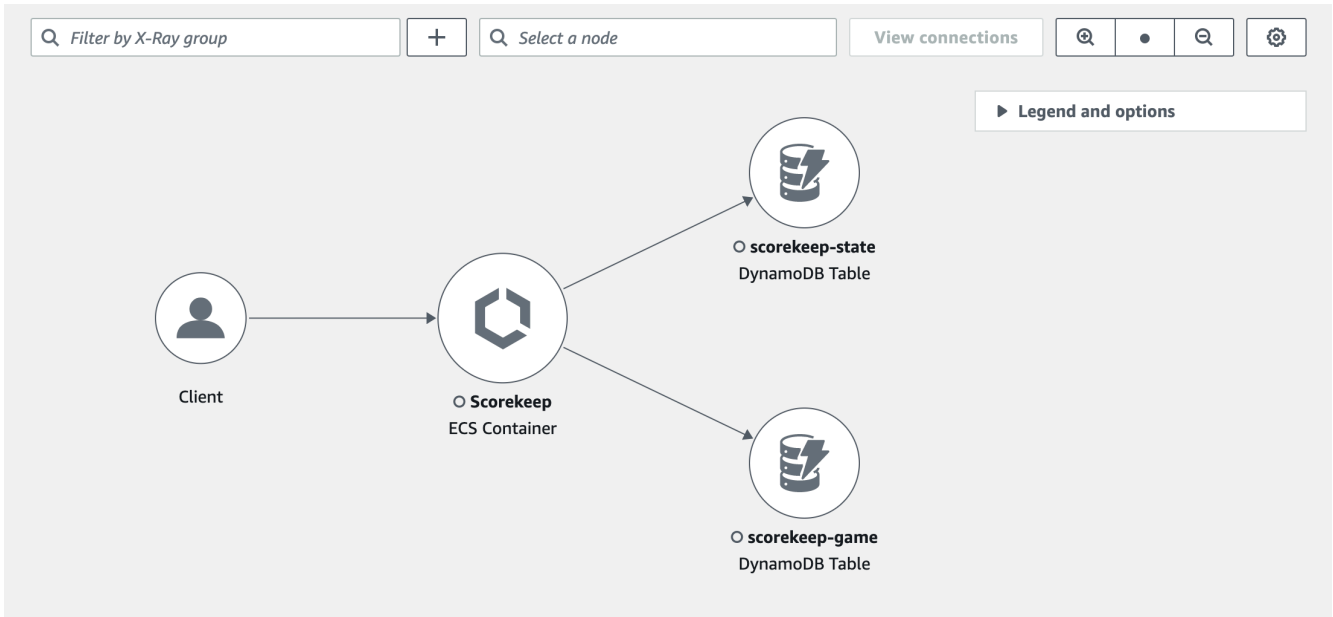
La carte de trace est une représentation visuelle des données de trace générées par vos applications. La carte montre les nœuds de service qui répondent aux demandes, les nœuds clients en amont qui représentent l'origine des demandes et les nœuds de service en aval qui représentent les services Web et les ressources utilisés par une application lors du traitement d'une demande.

La carte de trace affiche une vue connectée des traces dans les applications pilotées par des événements qui utilisent Amazon SQS et Lambda. Pour plus d'informations, consultez la section [Suivi des applications pilotées par des événements](#). La carte de suivi prend également en [charge le suivi entre comptes](#), en affichant les nœuds de plusieurs comptes sur une seule carte.

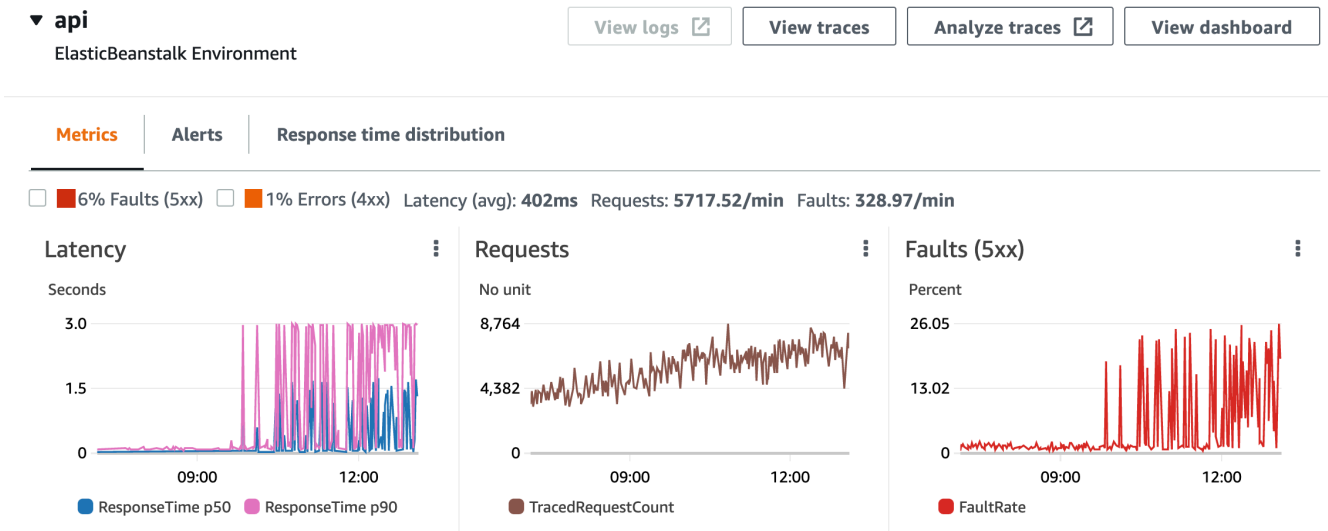
CloudWatch console

Pour afficher la carte de suivi dans la CloudWatch console

1. Ouvrez la [CloudWatch console](#). Choisissez Trace Map dans la section X-Ray Traces dans le volet de navigation de gauche.



2. Choisissez un nœud de service pour consulter les demandes pour ce nœud, ou un arc entre deux nœuds pour afficher les suivis des demandes qui ont circulé sur cette connexion.
3. Des informations supplémentaires sont affichées sous la carte de suivi, notamment des onglets pour les métriques, les alertes et la distribution des temps de réponse. Dans l'onglet Mesures, sélectionnez une plage dans chaque graphique pour accéder à une vue plus détaillée, ou choisissez les options Défauts ou Erreurs pour filtrer les traces. Dans l'onglet Distribution du temps de réponse, sélectionnez une plage dans le graphique pour filtrer les traces en fonction du temps de réponse.



- Affichez les traces en choisissant Afficher les traces, ou si un filtre a été appliqué, choisissez Afficher les traces filtrées.
- Choisissez Afficher les journaux pour voir CloudWatch les journaux associés au nœud sélectionné. Tous les nœuds de carte de trace ne prennent pas en charge l'affichage des journaux. Consultez les [CloudWatch journaux de résolution des problèmes](#) pour plus d'informations.

La carte de suivi indique les problèmes rencontrés au sein de chaque nœud en le décrivant en couleurs :

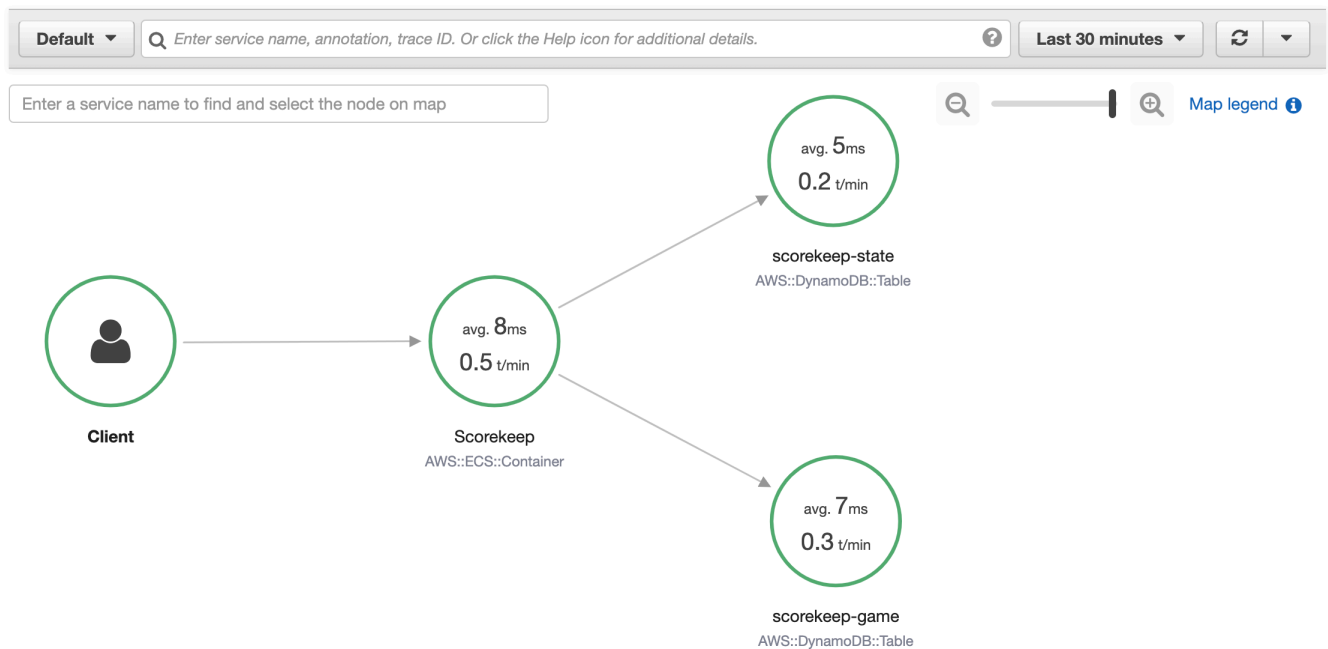
- Rouge pour les erreurs serveur (erreurs de type 500)
- Jaune pour les erreurs client (erreurs de type 400)
- Violet pour les erreurs de limitation (erreur 429, nombre de requêtes trop élevé)

Si votre carte de traçage est grande, utilisez les commandes à l'écran ou la souris pour zoomer ou dézoomer et déplacer la carte.

X-Ray console

Pour consulter la carte des services

- Ouvrez la [console X-Ray](#). La carte des services est affichée par défaut. Vous pouvez également sélectionner Service Map dans le volet de navigation de gauche.



2. Choisissez un nœud de service pour consulter les demandes pour ce nœud, ou un arc entre deux nœuds pour afficher les suivis des demandes qui ont circulé sur cette connexion.
3. Utilisez l'[histogramme](#) de distribution des réponses pour filtrer les traces par durée et sélectionnez les codes d'état pour lesquels vous souhaitez afficher les traces. Ensuite, choisissez View traces (Afficher les suivis) pour ouvrir la liste de suivi avec l'expression de filtre appliquée.

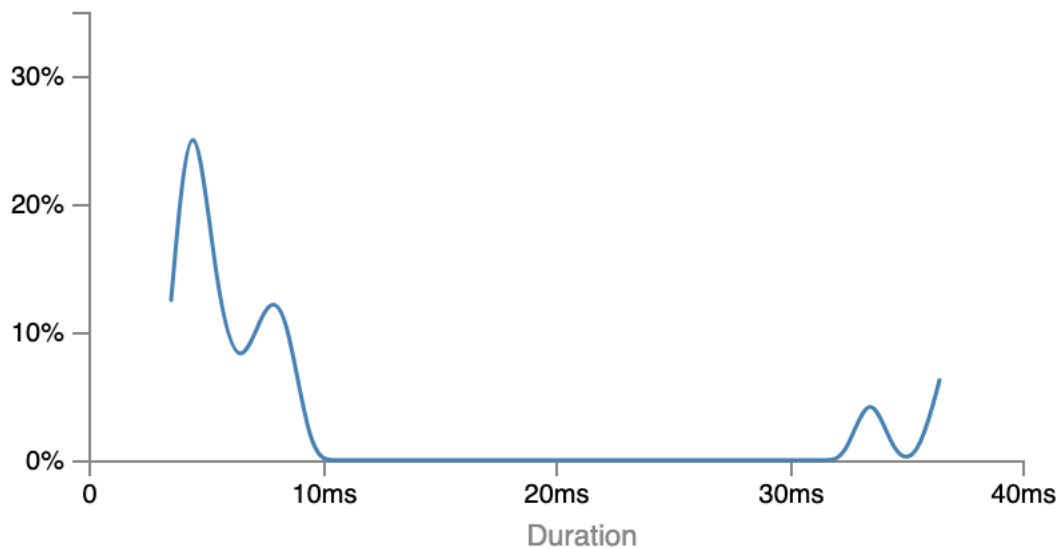
Service details ?

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 >

La cartographie des services indique l'état de chaque nœud en lui appliquant une couleur déterminée par le rapport entre le nombre d'appels réussis et le nombre d'erreurs ou d'échecs:

- Vert pour les appels réussis
- Rouge pour les erreurs serveur (erreurs de type 500)
- Jaune pour les erreurs client (erreurs de type 400)
- Violet pour les erreurs de limitation (erreur 429, nombre de requêtes trop élevé)

Si votre carte de service est grande, utilisez les commandes à l'écran ou la souris pour zoomer ou dézoomer et déplacer la carte.

Note

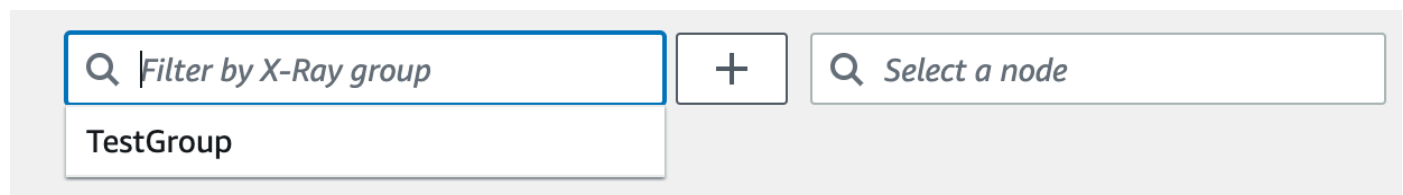
La carte de trace X-Ray peut afficher jusqu'à 10 000 nœuds. Dans de rares cas où le nombre total de nœuds de service dépasse cette limite, il est possible que vous receviez une erreur et que vous ne puissiez pas afficher une carte de suivi complète dans la console.

Filtrer la carte de trace par groupe

À l'aide d'une [expression de filtre](#), vous pouvez définir des critères selon lesquels inclure des traces au sein d'un groupe. Suivez les étapes ci-dessous pour afficher ensuite ce groupe spécifique sur la carte de trace.

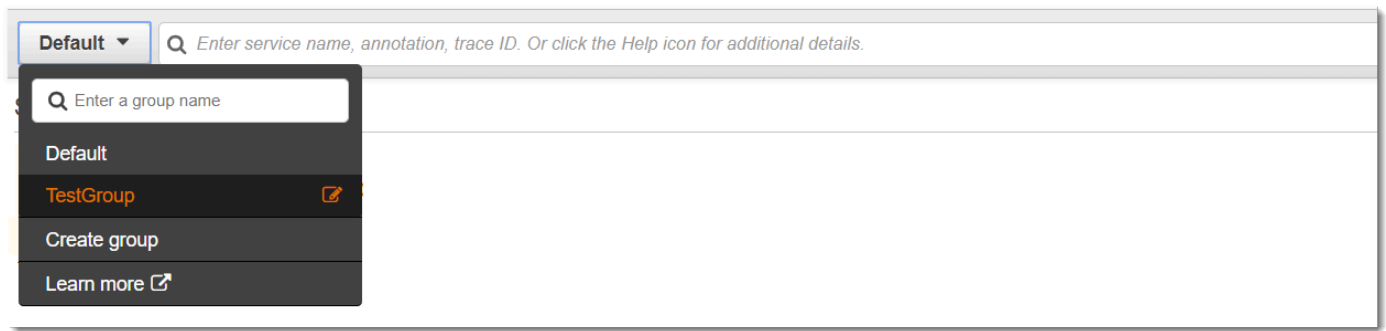
CloudWatch console

Choisissez un nom de groupe dans le filtre de groupe en haut à gauche de la carte de trace.



X-Ray console

Choisissez un nom de groupe dans le menu déroulant à gauche de la barre de recherche.



La carte des services sera désormais filtrée pour afficher les traces correspondant à l'expression du filtre du groupe sélectionné.

Tracez la légende et les options de la carte

La carte de trace inclut une légende et plusieurs options permettant de personnaliser l'affichage de la carte.

CloudWatch console

Choisissez le menu déroulant Légende et options en haut à droite de la carte. Choisissez ce qui est affiché dans les nœuds, notamment :

- Metrics affiche le temps de réponse moyen et le nombre de traces envoyées par minute pendant la période choisie.
- Nodes affiche l'icône de service dans chaque nœud.

Choisissez des paramètres de carte supplémentaires dans le volet Préférences, accessible via l'icône en forme de roue dentée en haut à droite de la carte. Ces paramètres incluent la sélection de la métrique utilisée pour déterminer la taille de chaque nœud et des canaris à afficher sur la carte.

X-Ray console

Affichez la légende de la carte de service en cliquant sur le lien Légende de la carte en haut à droite de la carte. Les options de carte de service peuvent être choisies en bas à droite de la carte de suivi, notamment :

- Les icônes de service activent ce qui est affiché dans chaque nœud, en affichant soit l'icône du service, soit le temps de réponse moyen et le nombre de traces envoyées par minute pendant la période choisie.
- Dimensionnement des nœuds : aucun ne définit la même taille pour tous les nœuds.
- Dimensionnement des nœuds : Health classe les nœuds en fonction du nombre de demandes concernées, y compris les erreurs, les défauts ou les demandes limitées.
- Dimensionnement des nœuds : le trafic taille les nœuds en fonction du nombre total de demandes.

Afficher les traces et les détails des traces

Utilisez la page Traces de la console X-Ray pour rechercher des traces par URL, code de réponse ou autres données dans le résumé des traces. Après avoir sélectionné une trace dans la liste des traces, la page Détails de la trace affiche une carte des nœuds de service associés à la trace sélectionnée et une chronologie des segments de trace.

Affichage des suivis

CloudWatch console

Pour afficher les traces dans la CloudWatch console

1. Connectez-vous à la CloudWatch console AWS Management Console et ouvrez-la à l'adresse <https://console.aws.amazon.com/cloudwatch/>.
2. Dans le volet de navigation de gauche, choisissez X-Ray Traces, puis Traces. Vous pouvez filtrer par groupe ou saisir une [expression de filtre](#). Cela filtre les traces affichées dans la section Traces au bas de la page.

Vous pouvez également utiliser la carte des services pour accéder à un nœud de service spécifique, puis afficher les traces. Cela ouvre la page Traces avec une requête déjà appliquée.

3. Affinez votre requête dans la section Affineurs de requêtes. Pour filtrer les traces en fonction d'un attribut commun, choisissez une option dans la flèche vers le bas située à côté de Affiner la requête par. Les options disponibles sont les suivantes :
 - Nœud : filtrez les traces par nœud de service.

- ARN de la ressource — Filtrez les traces en fonction d'une ressource associée à une trace. Ces ressources incluent par exemple une instance Amazon Elastic Compute Cloud (Amazon EC2), une AWS Lambda fonction ou une Amazon DynamoDB table.
- Utilisateur : filtrez les traces à l'aide d'un ID utilisateur.
- Message de cause première de l'erreur : filtrez les traces par cause première de l'erreur.
- URL : filtrez les traces en fonction du chemin d'URL utilisé par votre application.
- Code d'état HTTP : filtrez les traces en fonction du code d'état HTTP renvoyé par votre application. Vous pouvez spécifier un code de réponse personnalisé ou sélectionner l'une des options suivantes :
 - 200— La demande a été acceptée.
 - 401— La demande ne comportait pas d'informations d'authentification valides.
 - 403— La demande ne comportait pas d'autorisations valides.
 - 404— Le serveur n'a pas pu trouver la ressource demandée.
 - 500— Le serveur a rencontré une situation inattendue et a généré une erreur interne.

Choisissez une ou plusieurs entrées, puis cliquez sur Ajouter à la requête pour les ajouter à l'expression de filtre en haut de la page.

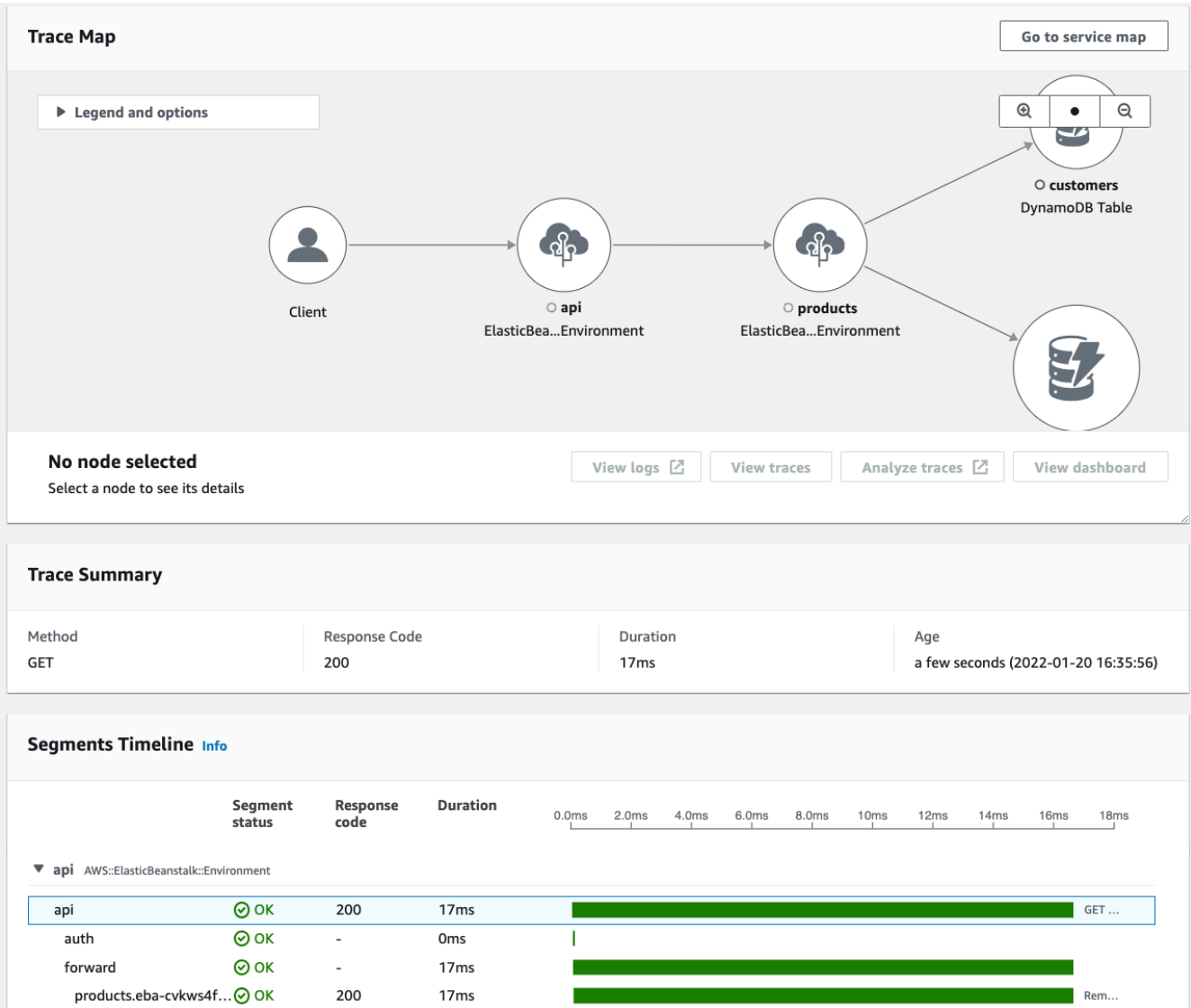
4. Pour rechercher une trace unique, entrez un [ID de trace](#) directement dans le champ de requête. Vous pouvez utiliser le format X-Ray ou le format World Wide Web Consortium (W3C). Par exemple, une trace créée à l'aide de [AWS Distro pour OpenTelemetry](#) est au format W3C.

 Note

Lorsque vous interrogez des traces créées avec un ID de trace au format W3C, la console affiche la trace correspondante au format X-Ray. Par exemple, si vous recherchez 4efaa4d1e8720b39541901950019ee5 au format W3C, la console affiche l'équivalent de X-Ray :1-4efaa4d-1e8720b39541901950019ee5.

5. Choisissez Exécuter la requête à tout moment pour afficher une liste des traces correspondantes dans la section Traces au bas de la page.
6. Pour afficher la page des détails du suivi pour un seul suivi, sélectionnez un ID de suivi dans la liste.

L'image suivante montre une carte de trace contenant les nœuds de service associés au tracé et les arêtes entre les nœuds représentant le chemin emprunté par les segments qui composent le tracé. Un résumé de trace suit la carte de trace. Le résumé contient des informations sur un exemple d'GETopération, son code de réponse, la durée d'exécution du suivi et l'âge de la demande. La chronologie des segments suit le résumé du suivi qui indique la durée des segments et sous-segments de suivi.



les applications pilotées par des événements, consultez. [Suivi des applications pilotées par des événements](#)

Les pages de détails des traces et des traces prennent également en [charge le suivi entre comptes](#), qui permet de répertorier les traces de plusieurs comptes dans la liste de traces et dans une seule carte de trace.

X-Ray console

Pour afficher les traces dans la console X-Ray

1. Ouvrez la page [Traces](#) dans la console X-Ray. Le panneau de présentation des traces affiche une liste de traces regroupées par fonctionnalités communes, notamment les causes profondes des erreurs, ResourceArn et. InstanceId
2. Pour sélectionner une fonction commune afin d'afficher un ensemble groupé de traces, déployez la flèche vers le bas à côté de Grouper par. L'illustration suivante présente une vue d'ensemble des traces regroupées par URL pour le [AWS X-Ray exemple d'application](#), ainsi qu'une liste des traces associées.

Trace overview

Group by: URL

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

Trace list (21)

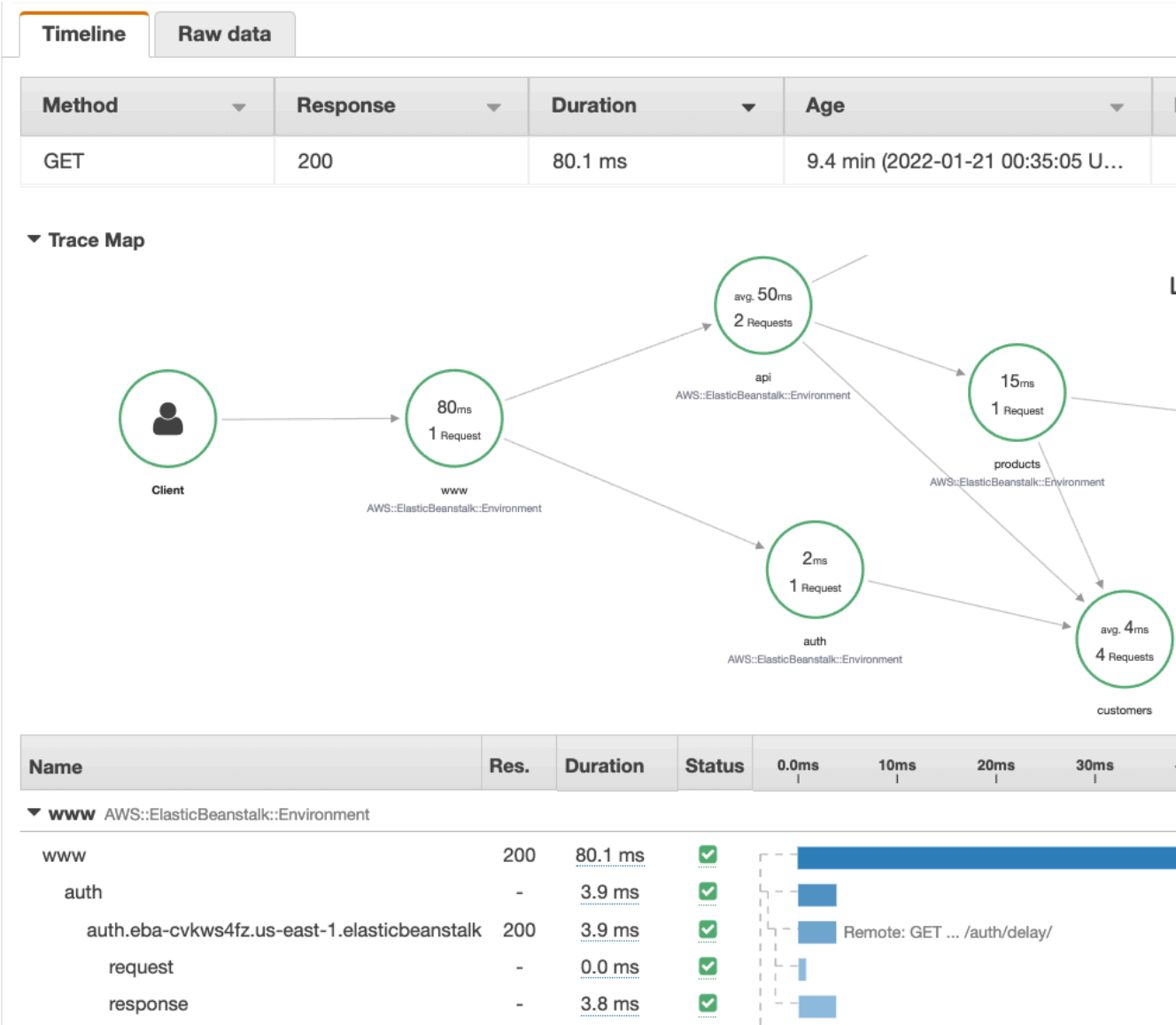
ID	Age	Method	Response	Response time	URL	Annotations
...f5f2df73	5.0 min	POST	200	391 ms	http://scorekeep.elasticbeanstalk.com/api/user	0
...cfe39980	5.0 min	PUT	200	33.0 ms	http://scorekeep.elasticbeanstalk.com/api/session/8N63LUQ6	0
...dd653e4c	5.0 min	POST	200	19.0 ms	http://scorekeep.elasticbeanstalk.com/api/session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

3. Choisissez l'ID d'une trace pour l'afficher dans la liste des traces. Vous pouvez également sélectionner Service map dans le volet de navigation pour afficher les traces d'un nœud de service spécifique. Vous pouvez ensuite afficher les traces associées à ce nœud.

L'onglet Chronologie indique le flux de demandes pour le suivi et inclut les éléments suivants :

- Une carte du chemin pour chaque segment de la trace.
- Le temps qu'il a fallu au segment pour atteindre un nœud de la carte de trace.
- Combien de demandes ont été adressées au nœud dans la carte de trace.

L'illustration suivante montre un exemple de carte de suivi associée à une GET demande envoyée à un exemple d'application. Les flèches indiquent le chemin emprunté par chaque segment pour terminer la demande. Les nœuds de service indiquent le nombre de demandes effectuées pendant la GET demande.



Pour plus d'informations sur l'onglet Chronologie, consultez la section Exploration de la chronologie de suivi suivante.

L'onglet Données brutes affiche des informations sur le tracé, ainsi que sur les segments et sous-segments qui le composent, sous JSON forme de format. Ces informations peuvent inclure les éléments suivants :

- Horodatages
- Unique IDs
- Ressources associées au segment ou au sous-segment
- La source, ou origine, du segment ou du sous-segment

- Informations supplémentaires sur la demande adressée à votre application, telles que la réponse d'une requête HTTP

Exploration de la chronologie des traces

La section Chronologie présente une hiérarchie de segments et de sous-segments à côté d'une barre horizontale qui correspond au temps qu'ils ont utilisé pour accomplir leurs tâches. La première entrée de la liste est le segment, qui représente toutes les données enregistrées par le service pour une seule demande. Les sous-segments sont mis en retrait et répertoriés après le segment. Les colonnes contiennent des informations sur chaque segment.

CloudWatch console

Dans la CloudWatch console, la chronologie des segments fournit les informations suivantes :

- La première colonne : répertorie les segments et sous-segments de la trace sélectionnée.
- La colonne État du segment : répertorie le résultat du statut de chaque segment et sous-segment.
- La colonne Code de réponse : répertorie un code d'état de réponse HTTP à une demande de navigateur effectuée par le segment ou le sous-segment, lorsqu'il est disponible.
- La colonne Durée : indique la durée du segment ou du sous-segment.
- La colonne Hosted in : répertorie l'espace de noms ou l'environnement dans lequel le segment ou le sous-segment est exécuté, le cas échéant. Pour plus d'informations, voir [Dimensions collectées et combinaisons de dimensions](#).
- La dernière colonne : affiche des barres horizontales qui correspondent à la durée pendant laquelle le segment ou le sous-segment s'est déroulé, par rapport aux autres segments ou sous-segments de la chronologie.

Pour regrouper la liste des segments et sous-segments par nœud de service, activez l'option Regrouper par nœuds.

X-Ray console

Sur la page des détails de la trace, choisissez l'onglet Chronologie pour voir la chronologie de chaque segment et sous-segment constituant une trace.

Dans la console X-Ray, la chronologie fournit les informations suivantes :

- La colonne Nom : répertorie les noms des segments et sous-segments de la trace.
- La colonne Res. : répertorie un code d'état de réponse HTTP à une demande de navigateur effectuée par le segment ou le sous-segment, lorsqu'il est disponible.
- La colonne Durée : indique la durée du segment ou du sous-segment.
- La colonne État : répertorie le résultat de l'état du segment ou du sous-segment.
- La dernière colonne : affiche des barres horizontales qui correspondent à la durée pendant laquelle le segment ou le sous-segment s'est déroulé, par rapport aux autres segments ou sous-segments de la chronologie.

Pour voir les données de trace brutes que la console utilise pour générer la chronologie, choisissez l'onglet Données brutes. Les données brutes vous fournissent des informations sur le tracé, ainsi que sur les segments et sous-segments qui le composent au JSON format. Ces informations peuvent inclure les éléments suivants :

- Horodatages
- Unique IDs
- Ressources associées au segment ou au sous-segment
- La source, ou origine, du segment ou du sous-segment
- Informations supplémentaires sur la demande adressée à votre application, telles que la réponse d'une requête HTTP.

Lorsque vous utilisez un AWS SDK instrumenté, HTTP, ou SQL client pour appeler des ressources externes, le SDK X-Ray enregistre automatiquement les sous-segments. Vous pouvez également utiliser le SDK X-Ray pour enregistrer des sous-segments personnalisés pour n'importe quelle fonction ou bloc de code. Les sous-segments supplémentaires enregistrés alors qu'un sous-segment personnalisé est ouvert deviennent les enfants du sous-segment personnalisé.

Affichage des détails d'un segment

Dans la chronologie du tracé, choisissez le nom d'un segment pour en afficher les détails.

Le panneau Détails du segment affiche les onglets Vue d'ensemble, Ressources, Annotations, Métadonnées, Exceptions et SQL. Les conditions suivantes s'appliquent :

- L'onglet Overview (Présentation) présente les informations relatives à la demande et à la réponse. Les informations incluent le nom, l'heure de début, l'heure de fin, la durée, l'URL de la demande,

le fonctionnement de la demande, le code de réponse à la demande et les erreurs et défauts éventuels.

- L'onglet Ressources d'un segment affiche des informations provenant du SDK X-Ray et des AWS ressources qui exécutent votre application. Utilisez les plug-ins Amazon EC2 ou Amazon ECS pour le SDK X-Ray afin d'enregistrer des informations sur les ressources spécifiques au service. AWS Elastic Beanstalk Pour plus d'informations sur les plug-ins, consultez la section Plug-ins de service dans [Configuration du SDK X-Ray pour Java](#).
- Les autres onglets affichent les annotations, les métadonnées et les exceptions enregistrées pour le segment. Les exceptions sont capturées automatiquement lorsqu'elles sont générées à partir d'une demande instrumentée. Les annotations et les métadonnées contiennent des informations supplémentaires que vous enregistrez à l'aide des opérations fournies par le SDK X-Ray. Pour ajouter des annotations ou des métadonnées à vos segments, utilisez le SDK X-Ray. Pour plus d'informations, consultez le lien spécifique à la langue répertorié sous Instrumentation de votre application avec. AWS X-Ray SDKs [Instrumentation de votre application pour AWS X-Ray](#)

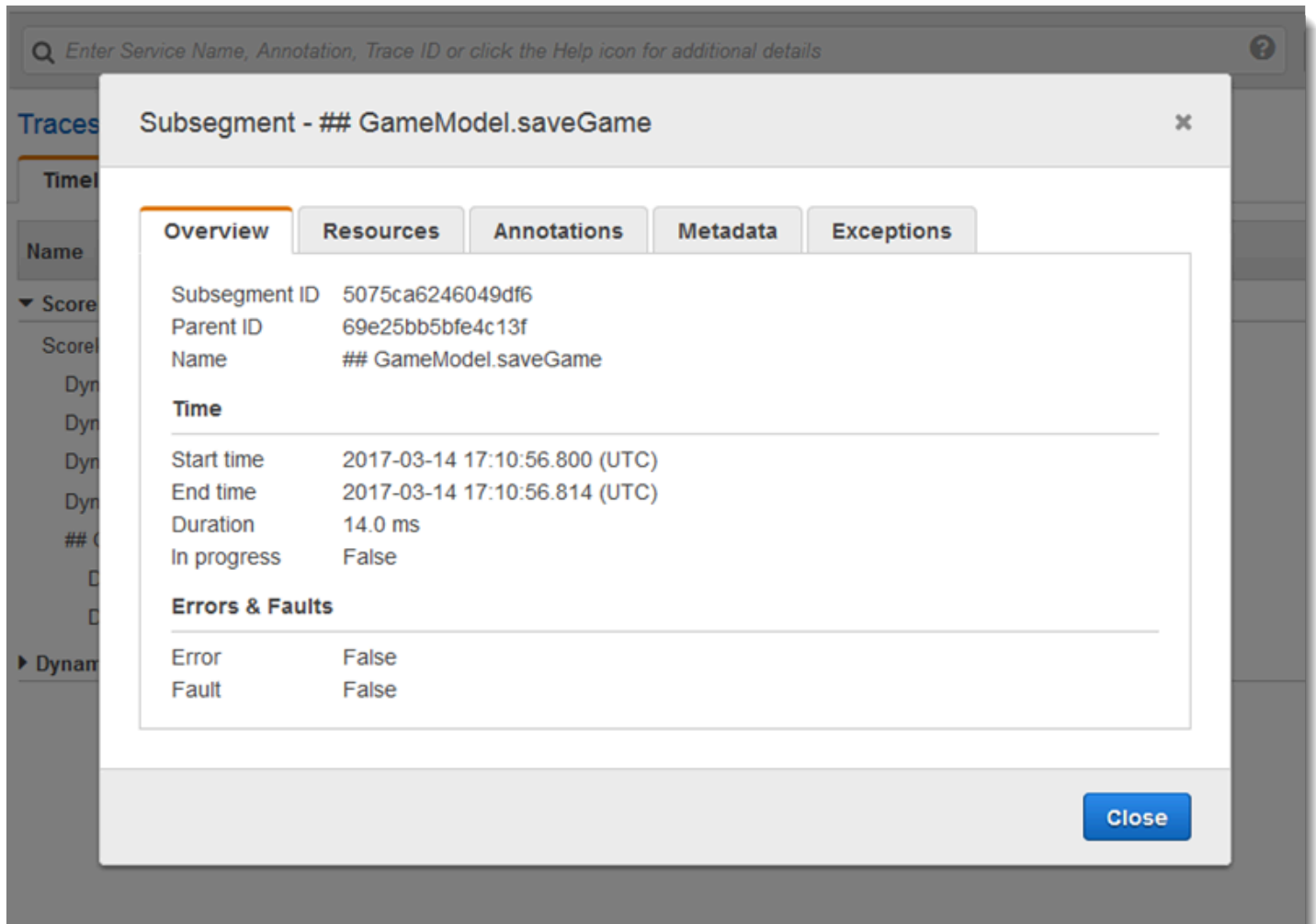
Affichage des détails d'un sous-segment

Dans la chronologie du suivi, choisissez le nom d'un sous-segment pour en afficher les détails :

- L'onglet Vue d'ensemble contient des informations sur la demande et la réponse. Cela inclut le nom, l'heure de début, l'heure de fin, la durée, la demande URL, le fonctionnement de la demande, le code de réponse à la demande et toutes les erreurs et défauts. Pour les sous-segments générés avec des clients instrumentés, l'onglet Overview (Présentation) contient des informations sur la demande et la réponse du point de vue de votre application.
- L'onglet Ressources d'un sous-segment affiche des informations détaillées sur les AWS ressources utilisées pour exécuter le sous-segment. Par exemple, l'onglet Ressources peut inclure un ARN de AWS Lambda fonction, des informations sur une table DynamoDB, toute opération appelée et un ID de demande.
- Les autres onglets affichent les annotations, les métadonnées et les exceptions enregistrées sur le sous-segment. Les exceptions sont capturées automatiquement lorsqu'elles sont générées à partir d'une demande instrumentée. Les annotations et les métadonnées contiennent des informations supplémentaires que vous enregistrez à l'aide des opérations fournies par le SDK X-Ray. Utilisez le SDK X-Ray pour ajouter des annotations ou des métadonnées à vos segments. Pour plus d'informations, consultez le lien spécifique à la langue répertorié sous Instrumentation de votre application avec. AWS X-Ray SDKs [Instrumentation de votre application pour AWS X-Ray](#)

Pour les sous-segments personnalisés, l'onglet Overview (Présentation) affiche le nom du sous-segment, que vous pouvez définir pour spécifier la zone de code ou la fonction que celui-ci enregistre. Pour plus d'informations, consultez le lien spécifique à la langue répertorié sous Instrumentation de votre application avec AWS X-Ray SDKs [Génération de sous-segments personnalisés avec le SDK X-Ray pour Java](#)

L'image suivante montre l'onglet Vue d'ensemble d'un sous-segment personnalisé. La vue d'ensemble contient l'ID du sous-segment, l'ID parent, le nom, les heures de début et de fin, la durée, le statut et les erreurs ou défauts.



L'onglet Métadonnées d'un sous-segment personnalisé contient des informations dans JSON format concernant les ressources utilisées par ce sous-segment.

Utilisation d'expressions de filtre

Utilisez des expressions de filtre pour afficher une carte de trace ou les traces d'une demande, d'un service, d'une connexion entre deux services (une périphérie) ou de demandes répondant à une condition spécifique. X-Ray fournit un langage d'expression de filtre pour filtrer les demandes, les services et les limites en fonction des données contenues dans les en-têtes des demandes, de l'état des réponses et des champs indexés des segments d'origine.

Lorsque vous choisissez une période de traces à afficher dans la console X-Ray, vous pouvez obtenir plus de résultats que ce que la console peut afficher. Dans le coin supérieur droit, la console affiche le nombre de suivis qu'elle a analysés et s'il y a d'autres suivis disponibles. Vous pouvez utiliser une expression de filtre pour limiter les résultats aux traces que vous souhaitez rechercher.

Rubriques

- [Filtrer les détails de l'expression](#)
- [Utilisation d'expressions de filtrage avec les groupes](#)
- [Syntaxe d'une expression de filtrage](#)
- [Mots-clés booléens](#)
- [Mots-clés valeur](#)
- [Mots-clés chaîne](#)
- [Mots-clés complexes](#)
- [fonction id](#)

Filtrer les détails de l'expression

Lorsque vous [choisissez un nœud dans la carte de trace](#), la console crée une expression de filtre en fonction du nom de service du nœud et des types d'erreurs présents en fonction de votre sélection. Pour trouver les suivis montrant les problèmes de performances ou concernant des demandes spécifiques, vous pouvez ajuster l'expression fournie par la console ou créer vos propres expressions. Si vous ajoutez des annotations avec le SDK X-Ray, vous pouvez également filtrer en fonction de la présence d'une clé d'annotation ou de la valeur d'une clé.

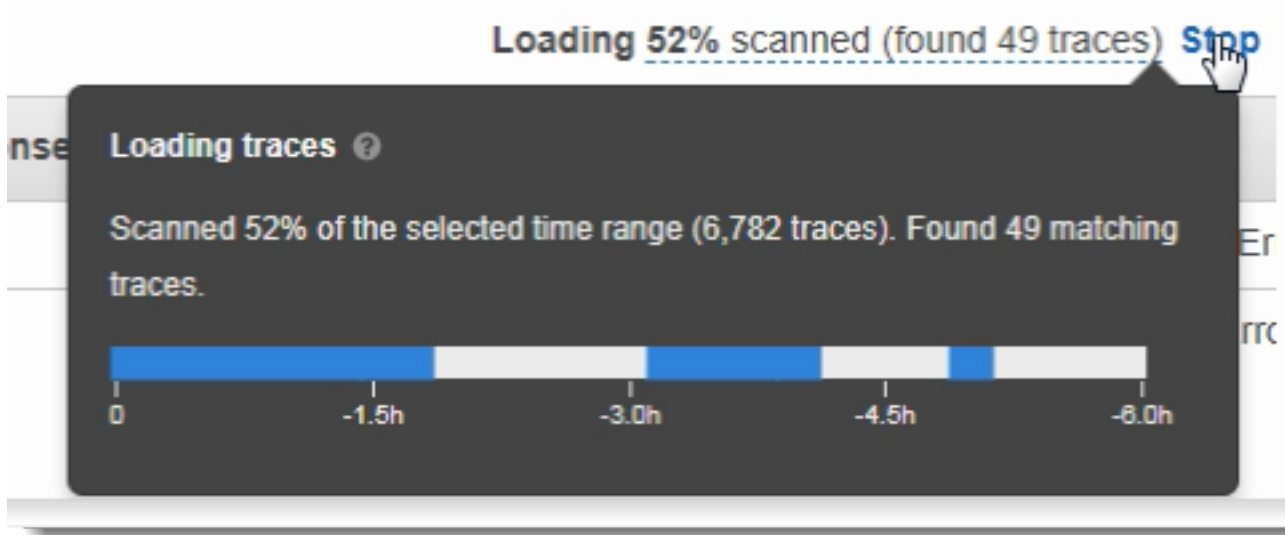
Note

Si vous choisissez une plage de temps relative dans la carte de trace et que vous choisissez un nœud, la console convertit la plage de temps en heure de début et de fin absolues.

Pour assurer que les suivis du nœud apparaissent dans les résultats de recherche et éviter d'analyser les heures auxquelles le nœud n'était pas actif, la plage de temps inclut uniquement les heures pendant lesquelles le nœud a envoyé des suivis. Pour effectuer une recherche par rapport à l'heure actuelle, vous pouvez revenir à une plage de temps relative dans la page des suivis et relancer l'analyse.

S'il y a toujours plus de résultats disponibles que la console ne peut en afficher, la console affiche le nombre de suivis correspondants et le nombre de suivis analysés. Le pourcentage affiché est le pourcentage de la période sélectionnée qui a été analysée. Pour vous assurer d'afficher tous les suivis correspondants représentés dans les résultats, réduisez encore votre expression de filtrage ou choisissez une période plus courte.

Pour obtenir les résultats les plus récents en premier, la console démarre l'analyse à la fin de la plage de temps et fonctionne en marche arrière. S'il y a un grand nombre de suivis, mais peu de résultats, la console fractionne la plage de temps en blocs qu'elle analyse en parallèle. La barre de progression indique les parties de la plage de temps qui ont été analysées.



Utilisation d'expressions de filtrage avec les groupes

Les groupes constituent un ensemble de suivis qui sont définis par une expression de filtre. Vous pouvez utiliser des groupes pour générer des graphiques de service supplémentaires et fournir CloudWatch des statistiques Amazon.

Les groupes sont identifiés par leur nom ou un Amazon Resource Name (ARN), et contiennent une expression de filtre. Le service compare les suivis entrants à l'expression, et les stocke en conséquence.

Vous pouvez créer et modifier des groupes en utilisant le menu déroulant à gauche de la barre de recherche d'expression de filtre.

 Note

Si le service rencontre une erreur en qualifiant un groupe, ce groupe n'est plus inclus dans le traitement des suivis entrants et une métrique d'erreur est enregistrée.

Pour plus d'informations sur les groupes, consultez [Configuration des groupes](#).

Syntaxe d'une expression de filtrage

Les expressions de filtrage peuvent contenir un mot-clé, un opérateur unaire ou binaire et une valeur de comparaison.

keyword operator value

Différents opérateurs sont disponibles pour les différents types de mot-clé. Par exemple, `responsetime` est un mot-clé valeur et peut être comparé aux opérateurs liés à des chiffres.

Exemple — demandes dont le temps de réponse était supérieur à 5 secondes

```
responsetime > 5
```

Vous pouvez combiner plusieurs expressions dans une expression composée avec les opérateurs AND ou OR.

Exemple — demandes dont la durée totale était de 5 à 8 secondes

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

Les mots-clés et opérateurs simples permettent uniquement de rechercher des problèmes au niveau des données de suivi. Si une erreur survient en aval, mais est gérée par votre application et n'est pas renvoyée à l'utilisateur, la recherche lancée sur `error` ne la trouvera pas.

Pour trouver les données de suivi concernant des problèmes en aval, vous pouvez utiliser les [mots-clés complexes](#) `service()` et `edge()`. Ces mots-clés vous permettent d'appliquer une expression de filtre à tous les nœuds en aval, à un seul nœud en aval ou à un arc entre deux nœuds. Pour encore plus de granularité, vous pouvez filtrer les services et les arcs par type avec [la fonction id\(\)](#).

Mots-clés booléens

Les valeurs des mots-clés booléens sont `true` ou `false`. Utilisez ces mots-clés pour trouver des données de suivi ayant généré des erreurs.

Mots-clés booléens

- `ok`— Le code d'état de la réponse était 2XX Success.
- `error`— Le code d'état de la réponse était 4XX Client Error.
- `throttle`— Le code d'état de la réponse était 429 trop de demandes.
- `fault`— Le code d'état de la réponse était une erreur de serveur 5XX.
- `partial`— La demande contient des segments incomplets.
- `inferred`— La demande contient des segments déduits.
- `first`— L'élément est le premier d'une liste énumérée.
- `last`— L'élément est le dernier d'une liste énumérée.
- `remote`— L'entité responsable est distante.
- `root`— Le service est le point d'entrée ou le segment racine d'une trace.

Les opérateurs booléens trouvent les segments dans lesquels la clé spécifiée est `true` ou `false`.

Opérateurs booléens

- `none` — L'expression est vraie si le mot clé est vrai.
- `!`— L'expression est vraie si le mot clé est faux.
- `=, !=` — Comparez la valeur du mot-clé à la chaîne `true` ou `false`. Ces opérateurs agissent de la même manière que les autres opérateurs mais sont plus explicites.

Exemple — le statut de la réponse est 2XX OK

```
ok
```

Exemple — le statut de la réponse n'est pas 2XX OK

```
!ok
```

Exemple — le statut de la réponse n'est pas 2XX OK

```
ok = false
```

Exemple — la dernière trace de panne énumérée porte le nom d'erreur « désérialiser »

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

Exemple — demandes avec des segments distants dont la couverture est supérieure à 0,7 et dont le nom du service est « traces »

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

Exemple — requêtes contenant des segments inférés dont le type de service est « AWS:DynamoDB »

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

Exemple — requêtes dont un segment porte le nom « plan de données » comme racine

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

Mots-clés valeur

Utilisez les mots-clés valeur pour rechercher des demandes avec un temps de réponse, une durée ou un statut de réponse spécifique.

Mots-clés valeur

- `responsetime`— Temps nécessaire au serveur pour envoyer une réponse.
- `duration`— Durée totale de la demande, y compris tous les appels en aval.
- `http.status`— Code d'état de la réponse.
- `index`— Position d'un élément dans une liste énumérée.

- **coverage**— Pourcentage décimal du temps de réponse de l'entité par rapport au temps de réponse du segment racine. Applicable uniquement aux entités de cause racine des temps de réponse.

Opérateurs numériques

Les mots-clés valeur utilisent l'égalité standard et les opérateurs de comparaison.

- **=, !=** — Le mot clé est égal ou non à une valeur numérique.
- **<, <=, >, >=** — Le mot clé est inférieur ou supérieur à une valeur numérique.

Exemple — le statut de la réponse n'est pas 200 OK

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

Exemple — demande dont la durée totale était de 5 à 8 secondes

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

Exemple — demandes traitées avec succès en moins de 3 secondes, y compris tous les appels en aval

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

Exemple — entité de liste énumérée dont l'indice est supérieur à 5

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

Exemple — demandes pour lesquelles la dernière entité dont la couverture est supérieure à 0,8

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

Mots-clés chaîne

Utilisez des mots-clés de chaîne pour rechercher des traces contenant du texte spécifique dans les en-têtes de demande ou un utilisateur IDs spécifique.

Mots-clés chaîne

- `http.url`— URL de demande.
- `http.method`— Méthode de demande.
- `http.useragent`— Demande une chaîne d'agent utilisateur.
- `http.clientip`— Adresse IP du demandeur.
- `user`— Valeur du champ utilisateur sur n'importe quel segment de la trace.
- `name`— Le nom d'un service ou d'une exception.
- `type`— Type de service
- `message`— Message d'exception.
- `availabilityzone`— Valeur du champ de zone de disponibilité sur n'importe quel segment de la trace.
- `instance.id`— Valeur du champ ID d'instance sur n'importe quel segment de la trace.
- `resource.arn`— Valeur du champ ARN de la ressource sur n'importe quel segment de la trace.

Les opérateurs chaîne trouvent des valeurs qui sont égales à un texte spécifique ou contiennent un texte spécifique. Ces valeurs doivent toujours être spécifiées entre guillemets.

Opérateurs de chaîne

- `=`, `!=` — Le mot clé est égal ou non à une valeur numérique.
- `CONTAINS`— Le mot clé contient une chaîne spécifique.
- `BEGINSWITH`, `ENDSWITH` — Le mot clé commence ou se termine par une chaîne spécifique.

Exemple — filtre `http.url`

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

Pour vérifier si un champ existe sur une donnée de suivi, quelle que soit sa valeur, vérifiez si elle contient la chaîne vide.

Exemple — filtre utilisateur

Trouvez toutes les traces avec l'utilisateur IDs.

```
user CONTAINS ""
```

Exemple — sélectionnez les traces dont la cause première du défaut inclut un service appelé « Auth »

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

Exemple — sélectionne les traces dont la cause première est le temps de réponse et dont le dernier service est de type DynamoDB

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

Exemple — sélectionne les traces dont la cause première est une erreur dont la dernière exception contient le message « accès refusé pour account_id : 1234567890 »

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

Mots-clés complexes

Utilisez les mots-clés complexes pour trouver des demandes en fonction du nom du service, du nom de l'arc ou d'une valeur d'annotation. Pour les services et les arcs, vous pouvez spécifier une expression de filtrage supplémentaire qui s'applique au service ou à l'arc. Pour les annotations, vous pouvez appliquer un filtre sur la valeur d'une annotation avec une clé spécifique en utilisant des opérateurs booléens, numériques ou de chaîne.

Mots-clés complexes

- `annotation[key]`— Valeur d'une annotation avec champ *key*. La valeur d'une annotation peut être un opérateur booléen, une valeur numérique ou une chaîne. Vous pouvez donc utiliser n'importe lequel de ces types d'opérateurs de comparaison. Vous pouvez utiliser ce mot clé en combinaison avec le edge mot clé `service` or. Une clé d'annotation contenant des points (points) doit être placée entre crochets (`[]`).
- `edge(source, destination) {filter}`— Connexion entre les services *source* et *destination*. Vous pouvez, si vous le souhaitez, utiliser des accolades pour englober une expression de filtrage à appliquer aux segments de cette connexion.
- `group.name / group.arn`— La valeur de l'expression de filtre d'un groupe, référencée par le nom ou l'ARN du groupe.

- `json`— Objet source JSON. Consultez [Getting data from AWS X-Ray](#) pour connaître les étapes de création d'entités JSON par programmation.
- `service(name) {filter}`— Service avec nom *name*. Vous pouvez, si vous le souhaitez, utiliser des accolades pour englober une expression de filtrage à appliquer aux segments créés par le service.

Utilisez le mot clé `service` pour trouver des traces pour les requêtes qui atteignent un certain nœud de votre carte de suivi.

Les opérateurs de mots clés complexes trouvent les segments dans lesquels la clé spécifiée a été définie ou non définie.

Opérateurs de mots clés complexes

- `none` — L'expression est vraie si le mot clé est défini. Si le mot clé est de type booléen, il sera évalué à la valeur booléenne.
- `!` — L'expression est vraie si le mot clé n'est pas défini. Si le mot clé est de type booléen, il sera évalué à la valeur booléenne.
- `=, !=` — Comparez la valeur du mot clé.
- `edge(source, destination) {filter}`— Connexion entre les services *source* et *destination*. Vous pouvez, si vous le souhaitez, utiliser des accolades pour englober une expression de filtrage à appliquer aux segments de cette connexion.
- `annotation[key]`— Valeur d'une annotation avec champ *key*. La valeur d'une annotation peut être un opérateur booléen, une valeur numérique ou une chaîne. Vous pouvez donc utiliser n'importe lequel de ces types d'opérateurs de comparaison. Vous pouvez utiliser ce mot clé en combinaison avec le `edge` mot clé `service` or.
- `json`— Objet source JSON. Consultez [Getting data from AWS X-Ray](#) pour connaître les étapes de création d'entités JSON par programmation.

Utilisez le mot clé `service` pour trouver des traces pour les requêtes qui atteignent un certain nœud de votre carte de suivi.

Exemple — Filtre de service

Demandes comprenant un appel vers `api.example.com` avec une anomalie (erreur de la série 500).


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

Vous pouvez exclure le nom du service pour appliquer une expression de filtre à tous les nœuds sur votre cartographie des services.

Exemple — filtre de service

Demandes à l'origine d'une erreur n'importe où sur votre carte de trace.

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

Le mot-clé `arc` applique une expression de filtre à une connexion entre deux nœuds.

Exemple — filtre de bord

Demande pour laquelle l'appel du service `api.example.com` à `backend.example.com` a échoué suite à une erreur.

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

Vous pouvez également utiliser l'opérateur `!` avec les mots-clés `service` et `arc` pour exclure un service ou un arc des résultats d'une autre expression de filtre.

Exemple — filtre de service et de demande

Demande pour laquelle l'URL commence par `http://api.example.com/` et inclut `/v2/` mais ne contacte pas un service nommé `api.example.com`.

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

Exemple — filtre de service et de temps de réponse

Trouvez des traces là où `http url` est défini et où le temps de réponse est supérieur à 2 secondes.

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

Pour les annotations, vous pouvez appeler toutes les traces où `annotation[key]` est défini ou utiliser les opérateurs de comparaison correspondant au type de valeur.

Exemple — annotation avec valeur de chaîne

Demandes avec une annotation nommée `gameid` avec une valeur de chaîne `"817DL6V0"`.

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

Exemple — l'annotation est définie

Demandes avec une annotation nommée `age set`.

```
annotation[age]
```

Exemple — l'annotation n'est pas définie

Demandes sans annotation nommées `age set`.

```
!annotation[age]
```

Exemple — annotation avec valeur numérique

Demandes avec un âge d'annotation avec une valeur numérique supérieure à 29.

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

Exemple — annotation en combinaison avec le service ou l'arête

```
service { annotation[request.id] = "917DL6V0" }
```

```
edge { source.annotation[request.id] = "916DL6V0" }
```

```
edge { destination.annotation[request.id] = "918DL6V0" }
```

Exemple — groupe avec utilisateur

Demandes où les traces rencontrent le filtre de `high_response_time` groupe (par exemple `responseTime > 3`) et où l'utilisateur s'appelle Alice.

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

Exemple — JSON avec entité de cause première

Requêtes avec entités de cause racine correspondantes.

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

fonction id

Lorsque vous fournissez un nom de service pour les mots-clés `service` ou `edge`, vous obtenez des résultats pour tous les nœuds correspondant à ce nom. Pour un filtrage plus précis, vous pouvez utiliser la fonction `id` pour spécifier un type de service en plus d'un nom et faire la distinction entre les nœuds portant le même nom.

Utilisez cette `account.id` fonction pour spécifier un compte particulier pour le service, lorsque vous consultez les traces de plusieurs comptes dans un compte de surveillance.

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

Vous pouvez utiliser la fonction `id` à la place d'un nom de service dans les filtres de service et d'arc.

```
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 }
```

Par exemple, AWS Lambda les fonctions génèrent deux nœuds dans la carte de trace : un pour l'invocation de la fonction et un pour le service Lambda. Les deux nœuds ont le même nom, mais des types différents. Un filtre de service standard trouvera des données de suivi pour les deux.

Exemple — filtre de service

Demandes qui incluent une erreur sur n'importe quel service nommé `random-name`.

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

Utilisez la fonction `id` pour limiter la recherche aux erreurs portant sur la fonction elle-même et exclure les erreurs du service.

Exemple — filtre de service avec fonction `id`

Demandes qui incluent une erreur sur un service nommé `random-name` de type `AWS::Lambda::Function`.

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

Pour rechercher des nœuds par type, vous pouvez également exclure complètement le nom.

Exemple — filtre de service avec fonction d'identification et type de service

Demandes qui incluent une erreur sur un service de type `AWS::Lambda::Function`.

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

Pour rechercher des nœuds pour un utilisateur en particulier Compte AWS, spécifiez un ID de compte.

Exemple — filtre de service avec fonction d'identification et identifiant de compte

Demandes qui incluent un service associé à un identifiant de compte spécifique `AWS::Lambda::Function`.

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

Traçabilité entre comptes

AWS X-Ray prend en charge l'observabilité entre comptes, vous permettant de surveiller et de dépanner les applications qui couvrent plusieurs comptes au sein d'une Région AWS. Vous pouvez facilement rechercher, visualiser et analyser vos indicateurs, journaux et traces dans tous les comptes associés, comme si vous travailliez sur un seul compte. Cela fournit une vue complète des demandes qui transitent par plusieurs comptes. Vous pouvez consulter les traces entre comptes sur la carte de suivi X-Ray et sur les pages de traces de la [CloudWatchconsole](#).

Les données d'observabilité partagées peuvent inclure l'un des types de télémétrie suivants :

- Métriques sur Amazon CloudWatch

- Groupes de journaux dans Amazon CloudWatch Logs
- Traces dans AWS X-Ray
- Applications dans Amazon CloudWatch Application Insights

Configurer l'observabilité entre comptes

Pour activer l'observabilité entre comptes, configurez un ou plusieurs comptes de AWS surveillance et associez-les à plusieurs comptes sources. Un compte de surveillance est une centrale Compte AWS qui permet de visualiser et d'interagir avec les données d'observabilité générées à partir des comptes sources. Un compte source est une personne Compte AWS qui génère des données d'observabilité pour les ressources qu'il contient.

Les comptes sources partagent leurs données d'observabilité avec les comptes de surveillance. Les traces sont copiées de chaque compte source vers un maximum de cinq comptes de surveillance. Les copies des traces des comptes sources vers le premier compte de surveillance sont gratuites. Les copies des traces envoyées à des comptes de surveillance supplémentaires sont facturées à chaque compte source, sur la base de la tarification standard. Pour plus d'informations, consultez les sections [AWS X-Ray Tarification](#) et [CloudWatch Tarification Amazon](#).

Pour créer des liens entre les comptes de surveillance et les comptes sources, utilisez la CloudWatch console ou les nouvelles commandes Observability Access Manager de l'API AWS CLI and. Pour plus d'informations, consultez la section [Observabilité inter-comptes de CloudWatch](#) (français non garanti).

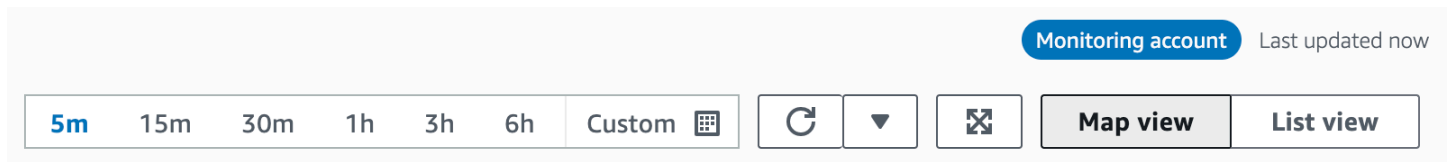
Note

Les tracés X-Ray sont facturés à l' Compte AWS endroit où ils sont reçus. Si une demande [échantillonnée](#) couvre plusieurs services Compte AWS, chaque compte enregistre une trace distincte, et toutes les traces partagent le même identifiant de trace. Pour en savoir plus sur la tarification de l'observabilité entre comptes, consultez les sections Tarification et [AWS X-Ray Tarification Amazon CloudWatch](#) .

Afficher les traces entre comptes

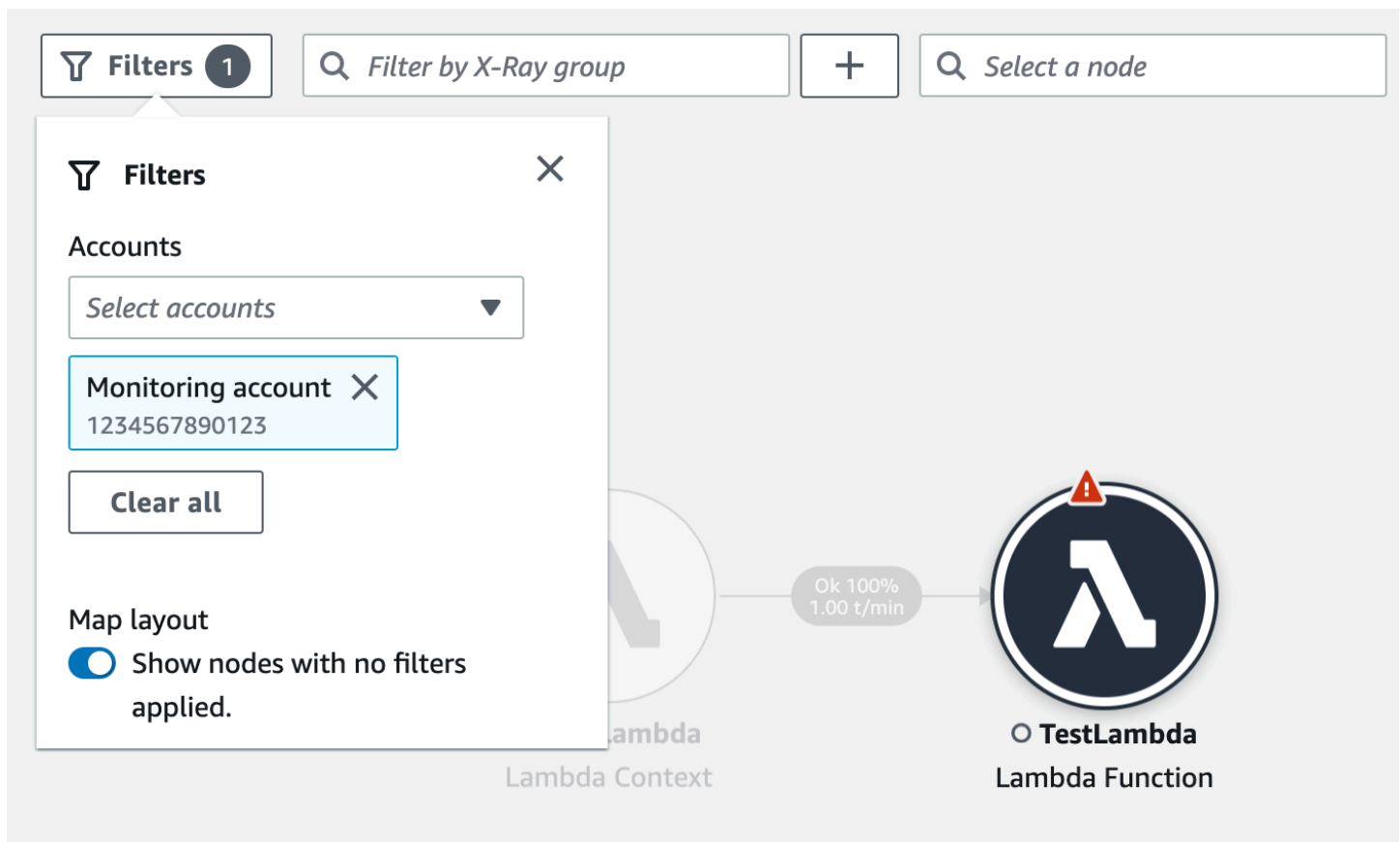
Les traces entre comptes sont affichées dans le compte de surveillance. Chaque compte source affiche uniquement les traces locales pour ce compte spécifique. Les sections suivantes supposent

que vous êtes connecté au compte de surveillance et que vous avez ouvert la CloudWatch console Amazon. Sur les pages de la carte de suivi et des traces, un badge de compte de surveillance est affiché dans le coin supérieur droit.



Carte de traçage

Dans la CloudWatch console, choisissez Trace Map sous X-Ray Traces dans le volet de navigation de gauche. Par défaut, la carte de suivi affiche les nœuds de tous les comptes sources qui envoient des traces au compte de surveillance, ainsi que les nœuds du compte de surveillance lui-même. Sur la carte de trace, choisissez Filtres en haut à gauche pour filtrer la carte de trace à l'aide de la liste déroulante Comptes. Une fois qu'un filtre de compte est appliqué, les nœuds de service des comptes qui ne correspondent pas au filtre actuel sont grisés.



Lorsque vous choisissez un nœud de service, le volet de détails du nœud inclut l'ID de compte et le libellé du service.

▼ **TestLambda**

Lambda Function

Account: Monitoring account (1234567890123)

View logs

View traces

Analyze traces

View dashboard

Metrics

Alerts

Response time distribution

Dans le coin supérieur droit de la carte de trace, choisissez Affichage en liste pour afficher la liste des nœuds de service. La liste des nœuds de service inclut les services du compte de surveillance et tous les comptes sources configurés. Filtrez la liste des nœuds par étiquette de compte ou identifiant de compte en les choisissant dans le filtre Nœuds.

Nodes (2)

Q Account id =

Use: "Account id = "

Values

Account id = 461265027466

Alarms ▼

Latency (avg) ▼

Faults (5xx) ▼

⚠ 1

13ms

0.00/min

Suivis

Consultez les détails des traces couvrant plusieurs comptes en ouvrant la CloudWatch console depuis le compte de surveillance et en choisissant Traces sous X-Ray Traces dans le volet de navigation de gauche. Vous pouvez également ouvrir cette page en choisissant un nœud dans la X-Ray Trace Map, puis en choisissant View traces dans le volet des détails du nœud.

La page Traces permet d'effectuer des requêtes par identifiant de compte. Pour commencer, [entrez une requête](#) qui inclut un ou plusieurs comptes IDs. L'exemple suivant demande des traces transmises par le biais de l'ID de compte X ou Y :

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

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](#).

Q Filter by X-Ray group

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

Run query

5 traces retrieved

Affinez votre requête par compte. Sélectionnez un ou plusieurs comptes dans la liste, puis cliquez sur Ajouter à la requête.

▼ Query refiners

Refine query by Account ▼

1 selected

Add to query

Select rows to filter traces

Find Account name and ID

< 1 >



Account name and ID



Monitoring account (1234567890123)

Détails du tracé

Consultez les détails d'une trace en la sélectionnant dans la liste Traces en bas de la page Traces. Les détails du suivi s'affichent, y compris une carte des détails du suivi avec les nœuds de service de tous les comptes traversés par le suivi. Choisissez un nœud de service spécifique pour voir le compte correspondant.

La section Chronologie des segments affiche les détails du compte pour chaque segment de la chronologie.

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

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

Suivi des applications pilotées par des événements

AWS X-Ray prend en charge le suivi des applications pilotées par des événements à l'aide d'Amazon SQS et. AWS Lambda Utilisez la CloudWatch console pour voir une vue connectée de chaque demande mise en file d'attente avec Amazon SQS et traitée par une ou plusieurs fonctions Lambda. Les traces provenant des producteurs de messages en amont sont automatiquement liées aux traces provenant des nœuds consommateurs Lambda en aval, créant ainsi une end-to-end vue de l'application.

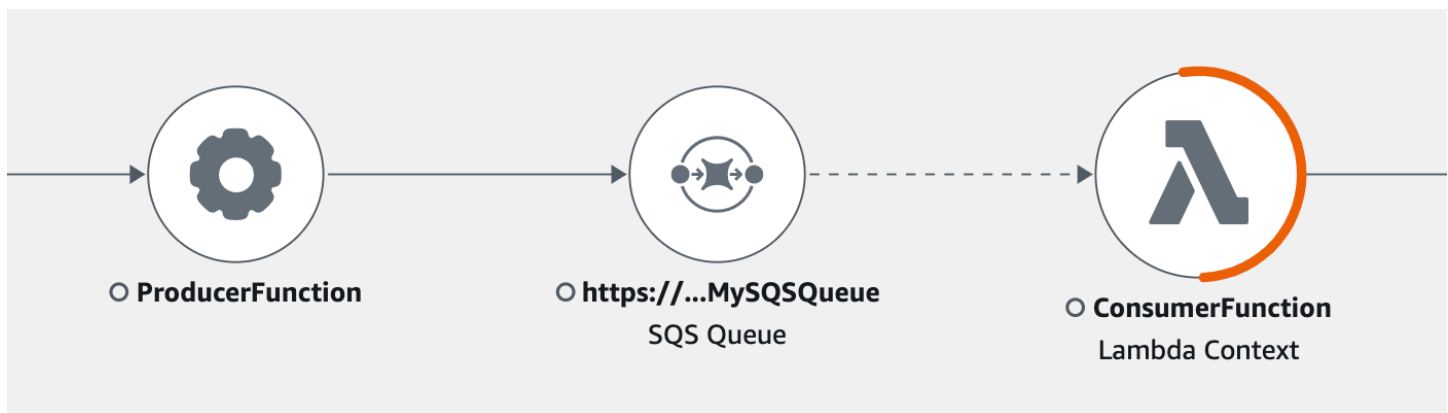
Note

Chaque segment de trace peut être lié à un maximum de 20 traces, tandis qu'une trace peut inclure un maximum de 100 liens. Dans certains scénarios, l'association de traces

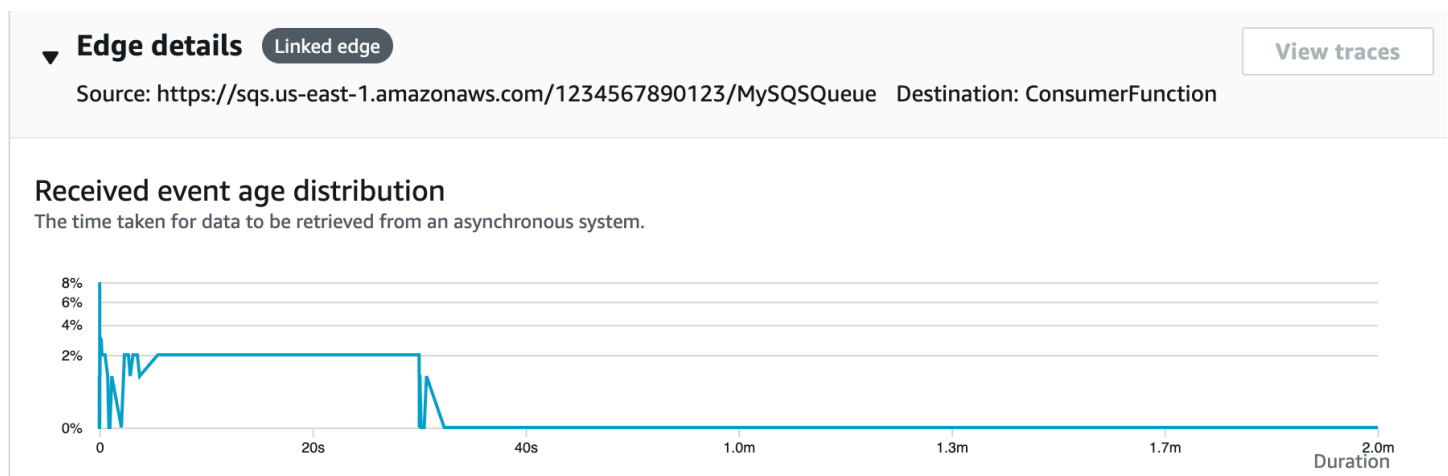
supplémentaires peut entraîner le dépassement de la [taille maximale du document de trace](#), ce qui peut entraîner une trace incomplète. Cela peut se produire, par exemple, lorsqu'une fonction Lambda dont le suivi est activé envoie de nombreux messages SQS à une file d'attente en un seul appel. Si vous rencontrez ce problème, une solution d'atténuation utilisant le X-Ray est disponible SDKs. Consultez le SDK X-Ray pour [Java](#), [Node.js](#), [Python](#), [Go](#) ou [.NET](#) pour plus d'informations.

Afficher les traces liées dans la carte des traces

Utilisez la page Trace Map de la [CloudWatchconsole](#) pour afficher une carte de suivi contenant les traces des créateurs de messages associées aux traces des consommateurs Lambda. Ces liens sont affichés avec un bord en pointillé qui connecte le nœud Amazon SQS aux nœuds consommateurs Lambda en aval.



Sélectionnez un bord en pointillé pour afficher un histogramme de l'âge de l'événement reçu, qui indique la répartition de l'âge de l'événement lorsqu'il est reçu par les consommateurs. L'âge est calculé chaque fois qu'un événement est reçu.



Afficher les détails du suivi lié

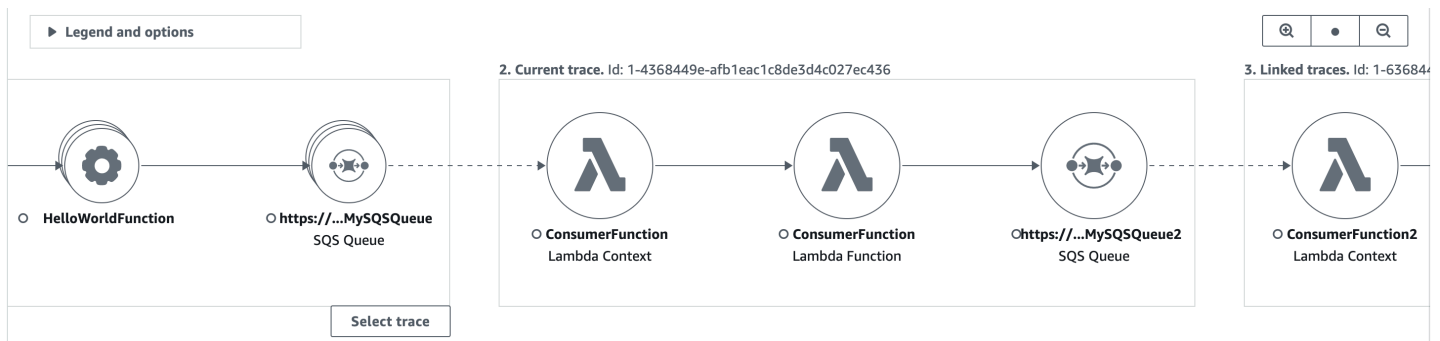
Consultez les informations de suivi envoyées par un producteur de messages, une file d'attente Amazon SQS ou un client Lambda :

1. Utilisez la carte de suivi pour sélectionner un nœud producteur de messages, Amazon SQS ou consommateur Lambda.
2. Choisissez Afficher les traces dans le volet des détails du nœud pour afficher la liste des traces. Vous pouvez également accéder directement à la page Traces dans la CloudWatch console.
3. Choisissez une trace spécifique dans la liste pour ouvrir la page des détails de la trace. La page des détails de la trace affiche un message lorsque la trace sélectionnée fait partie d'un ensemble de traces liées.

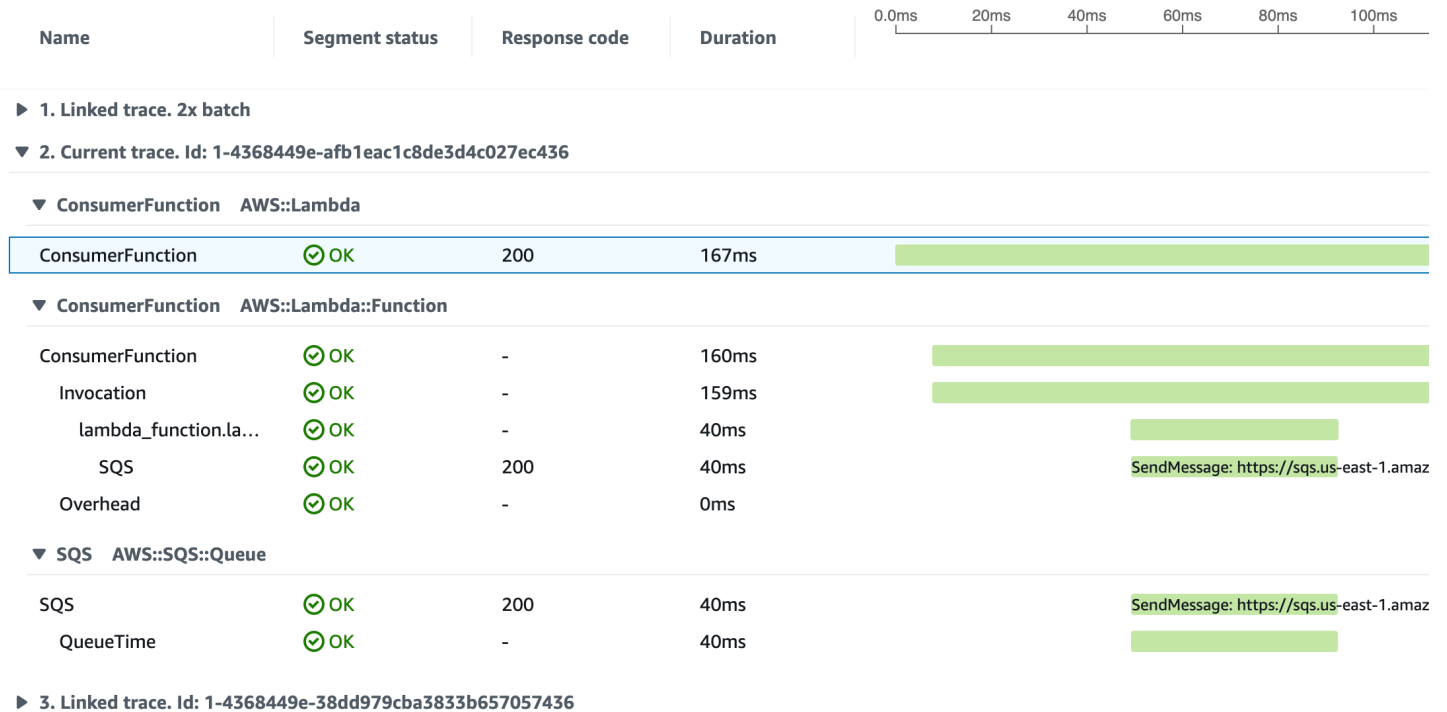
CloudWatch > Traces > Trace 1-4368449e-afb1eac1c8de3d4c027ec436

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

La carte des détails de la trace affiche la trace actuelle, ainsi que les traces liées en amont et en aval, chacune d'elles étant contenue dans un cadre indiquant les limites de chaque trace. Si la trace actuellement sélectionnée est liée à plusieurs traces en amont ou en aval, les nœuds des traces liées en amont ou en aval sont empilés et un bouton Sélectionner une trace s'affiche.



Sous la carte des détails du tracé, une chronologie des segments de trace s'affiche, y compris les traces liées en amont et en aval. S'il existe plusieurs traces liées en amont ou en aval, les détails de leurs segments ne peuvent pas être affichés. Pour afficher les détails d'un segment pour une seule trace au sein d'un ensemble de traces liées, [sélectionnez une seule trace](#) comme décrit ci-dessous.

Segments Timeline [Info](#)

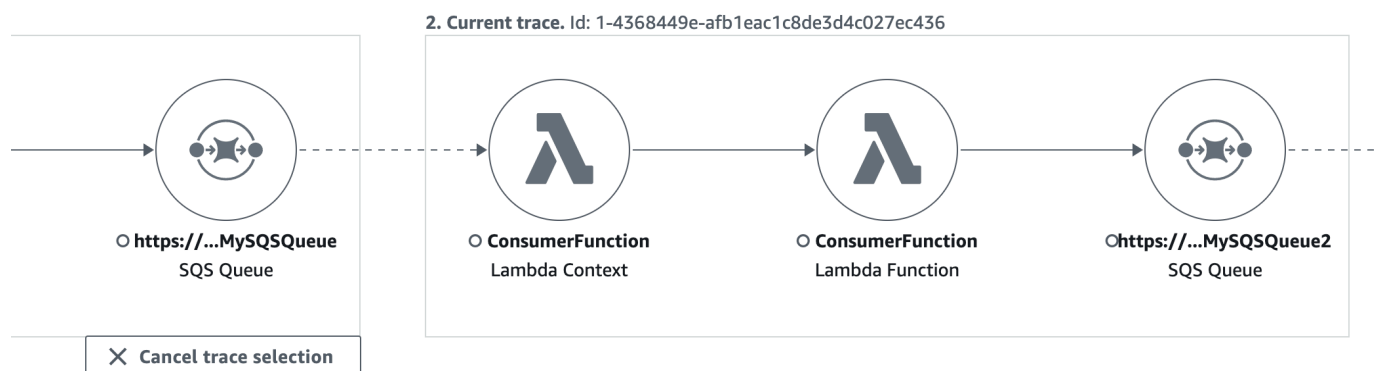
Sélectionnez une seule trace parmi un ensemble de traces liées

Filtrez un ensemble de traces liées en une seule trace, pour voir les détails des segments dans la chronologie.

1. Choisissez Sélectionner une trace sous les traces liées sur la carte des détails de la trace. La liste des traces s'affiche.

Traces (2)				
Start typing to filter trace list				
	ID	Trace status	Timestamp	Response code
☒	...3fd6e9600d58fea82597e9af	✓ OK	11.7min (2022-11-06 15:34:54)	200
☐	...223d41cc17bae4a5394423a0	✓ OK	11.7min (2022-11-06 15:34:54)	200

2. Sélectionnez le bouton radio à côté d'une trace pour l'afficher dans la carte des détails de la trace.
3. Choisissez Annuler la sélection de traces pour afficher l'ensemble complet des traces liées.



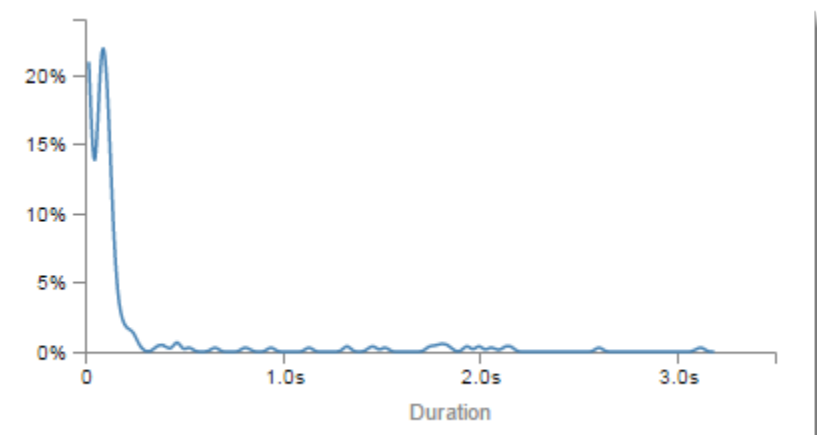
Utilisation d'histogrammes de latence

Lorsque vous sélectionnez un nœud ou une arête sur une [carte de AWS X-Ray traçage](#), la console X-Ray affiche un histogramme de distribution de latence.

Latence

La latence correspond au temps écoulé entre le moment où une demande commence et celui où elle prend fin. Un histogramme montre la répartition des latences. Il indique la durée sur l'axe X et le pourcentage de demandes correspondant à chaque durée sur l'axe Y.

Cet histogramme montre un service qui répond à la plupart des demandes en moins de 300 ms. Un faible pourcentage des demandes prennent jusqu'à 2 secondes et quelques valeurs hors norme peuvent prendre un peu plus de temps.



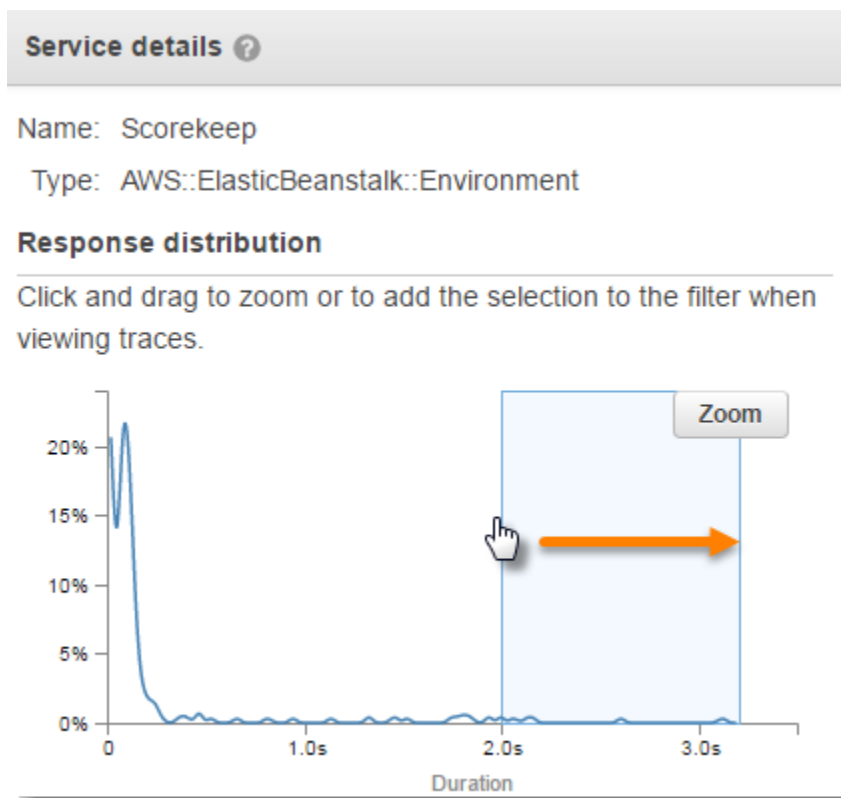
Interprétation des détails du service

Les histogrammes des services et des arcs offrent une représentation visuelle des latences du point de vue d'un service ou d'un demandeur.

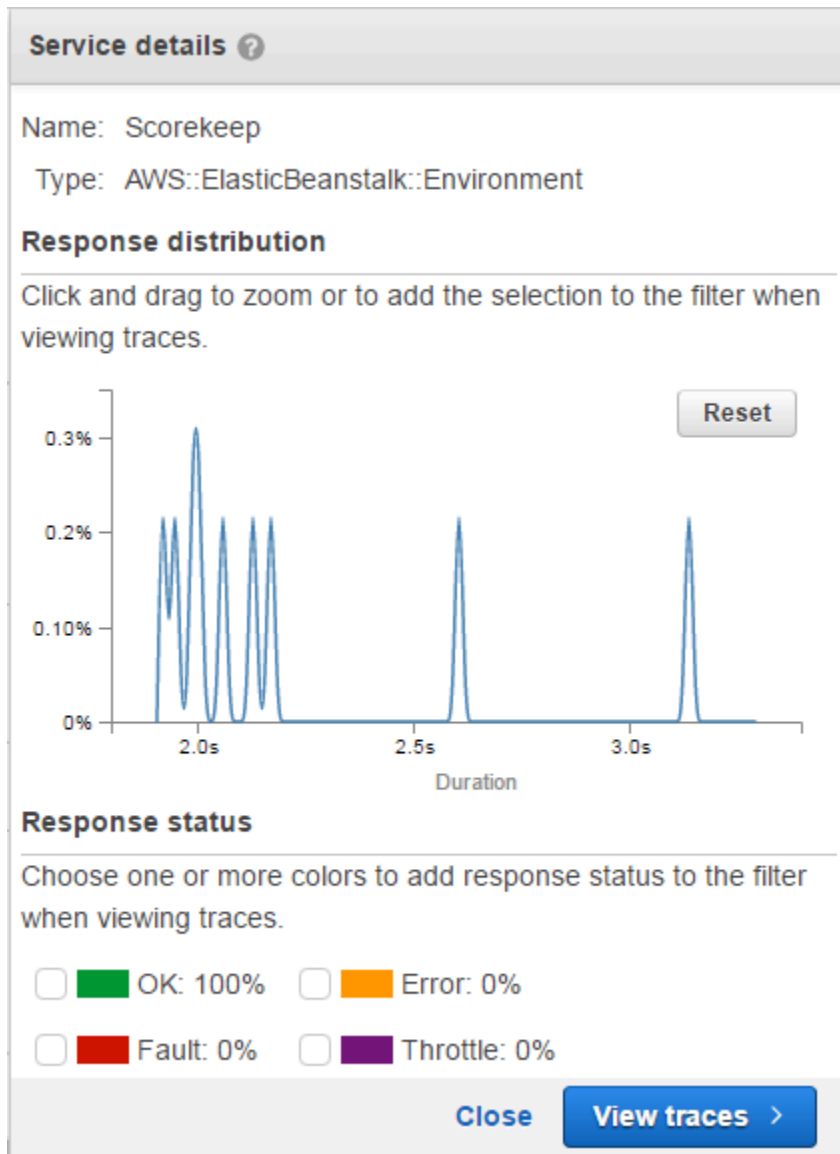
- Choisissez un nœud de service en cliquant sur le cercle. X-Ray affiche un histogramme des demandes traitées par le service. Les latences sont celles enregistrées par le service et n'incluent aucune latence réseau entre le service et le demandeur.
- Choisissez une arête en cliquant sur le bout de la ligne ou de la flèche située entre deux services. X-Ray affiche un histogramme des demandes du demandeur qui ont été traitées par le service en aval. Les latences sont celles enregistrées par le demandeur et incluent les latences au niveau de la connexion réseau entre les deux services.

Pour interpréter l'histogramme du panneau Service details, vous pouvez rechercher les valeurs les plus extrêmes. Ces valeurs hors norme peuvent être considérées comme des pics dans l'histogramme. Vous pouvez alors consulter les suivis d'une zone donnée pour voir ce qu'il se passe.

Pour afficher les suivis filtrés par latence, sélectionnez une plage dans l'histogramme. Cliquez à l'endroit où vous souhaitez commencer la sélection et faites glisser la souris de la gauche vers la droite pour mettre en surbrillance une plage de latences à inclure dans le filtre de suivi.



Une fois que vous avez sélectionné une plage, vous pouvez choisir Zoom pour afficher uniquement une certaine partie de l'histogramme et affiner votre sélection.



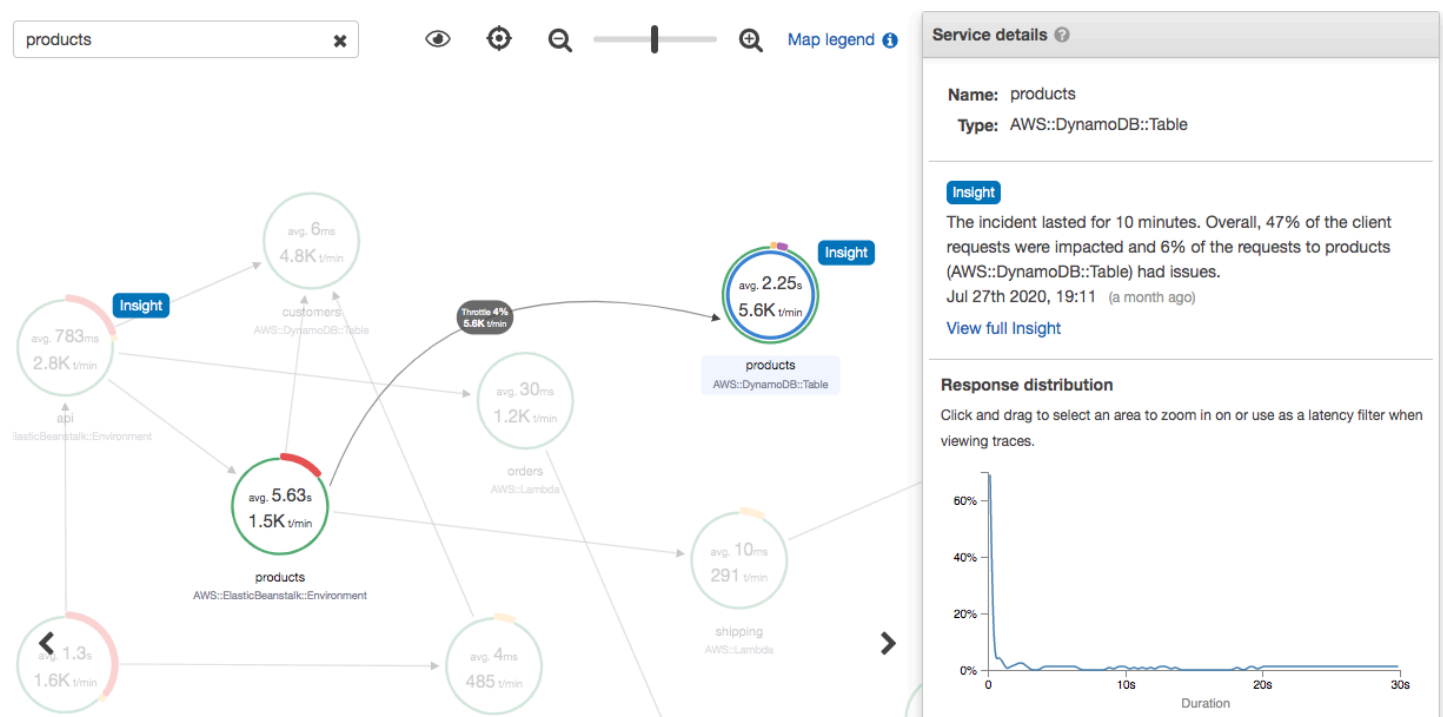
Une fois que vous avez défini la zone qui vous intéresse, choisissez View traces.

Utilisation des informations issues de X-Ray

AWS X-Ray analyse en permanence les données de suivi de votre compte afin d'identifier les problèmes émergents dans vos applications. Lorsque les taux de défaillance dépassent la plage prévue, cela crée un aperçu qui enregistre le problème et suit son impact jusqu'à ce qu'il soit résolu. Grâce à Insights, vous pouvez :

- Identifiez l'endroit où les problèmes se produisent dans votre application, la cause première du problème et l'impact associé. L'analyse d'impact fournie par Insights vous permet de déterminer la gravité et la priorité d'un problème.
- Recevez des notifications lorsque le problème évolue au fil du temps. Les notifications Insights peuvent être intégrées à votre solution de surveillance et d'alerte à l'aide d'Amazon EventBridge. Cette intégration vous permet d'envoyer des e-mails ou des alertes automatisés en fonction de la gravité du problème.

La console X-Ray identifie les nœuds présentant des incidents en cours sur la carte de trace. Pour voir un résumé des informations, choisissez le nœud concerné. Vous pouvez également afficher et filtrer les informations en choisissant Insights dans le volet de navigation de gauche.



X-Ray crée un aperçu lorsqu'il détecte une anomalie dans un ou plusieurs nœuds de la carte des services. Le service utilise une modélisation statistique pour prévoir les taux de défaillance attendus des services de votre application. Dans l'exemple précédent, l'anomalie est une augmentation du nombre de défauts provenant de AWS Elastic Beanstalk. Le serveur Elastic Beanstalk a connu plusieurs délais d'attente pour les appels d'API, ce qui a provoqué une anomalie dans les nœuds en aval.

Activez les informations dans la console X-Ray

Insights doit être activé pour chaque groupe avec lequel vous souhaitez utiliser les fonctionnalités Insights. Vous pouvez activer les informations depuis la page Groupes.

1. Ouvrez la [console X-Ray](#).
2. Sélectionnez un groupe existant ou créez-en un nouveau en choisissant Create group, puis en sélectionnant Enable Insights. Pour plus d'informations sur la configuration des groupes dans la console X-Ray, consultez [Configuration des groupes](#).
3. Dans le volet de navigation de gauche, choisissez Insights, puis choisissez un aperçu à afficher.

From

2021-01-18 20:00

To

2021-01-20 11:00

Group

Default

State

All

Apply filter

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. Closed Fault	2 minutes 58 seconds	api (AWS::ElasticBeanstalk::Envir...)	www (AWS::ElasticBeanstalk::Envir...) api (AWS::ElasticBeanstalk::Envir...)	Default	Jan 19th 2021, 19:02

Note

X-Ray utilise GetInsightSummaries, GetInsight GetInsightEvents, et des opérations GetInsightImpactGraph d'API pour récupérer des données à partir d'informations. Pour consulter les informations, utilisez la stratégie gérée par AWSXray ReadOnlyAccess IAM ou ajoutez la politique personnalisée suivante à votre rôle IAM :

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



```
}  
]
```

Pour de plus amples informations, veuillez consulter [Comment AWS X-Ray fonctionne avec IAM](#).

Activez les notifications d'informations

Avec les notifications d'informations, une notification est créée pour chaque événement d'information, par exemple lorsqu'une information est créée, change de manière significative ou est fermée. Les clients peuvent recevoir ces notifications par le biais d' EventBridge événements Amazon et utiliser des règles conditionnelles pour effectuer des actions telles que la notification SNS, l'invocation Lambda, la publication de messages dans une file d'attente SQS ou l'un des supports cibles. EventBridge Les notifications Insights sont émises dans la mesure du possible, mais ne sont pas garanties. Pour plus d'informations sur les cibles, consultez [Amazon EventBridge Targets](#).

Vous pouvez activer les notifications d'informations pour tout groupe activé sur la page Groupes.

Pour activer les notifications pour un groupe X-Ray

1. Ouvrez la [console X-Ray](#).
2. Sélectionnez un groupe existant ou créez-en un nouveau en choisissant Créer un groupe, assurez-vous que l'option Activer les informations est sélectionnée, puis sélectionnez Activer les notifications. Pour plus d'informations sur la configuration des groupes dans la console X-Ray, consultez [Configuration des groupes](#).

Pour configurer les règles EventBridge conditionnelles d'Amazon

1. Ouvrez la [EventBridge console Amazon](#).
2. Accédez à Règles dans la barre de navigation de gauche, puis choisissez Créer une règle.
3. Donnez un nom et une description à la règle.
4. Choisissez Modèle d'événement, puis choisissez Modèle personnalisé. Fournissez un modèle contenant "source": ["aws.xray"] et "detail-type": ["AWS X-Ray Insight Update"]. Voici quelques exemples de modèles possibles.

- Modèle d'événements correspondant à tous les événements entrants provenant de X-Ray Insights :

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

- Modèle d'événement correspondant à une valeur spécifiée **state** et **category** :

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

5. Sélectionnez et configurez les cibles que vous souhaitez appeler lorsqu'un événement répond à cette règle.
6. (Facultatif) Fournissez des balises pour identifier et sélectionner plus facilement cette règle.
7. Sélectionnez Create (Créer).

Note

Les notifications X-Ray Insights envoient des événements à Amazon EventBridge, qui ne prend actuellement pas en charge les clés gérées par les clients. Pour de plus amples informations, veuillez consulter [Protection des données dans AWS X-Ray](#).

Aperçu d'Insight

La page de présentation d'un aperçu tente de répondre à trois questions clés :

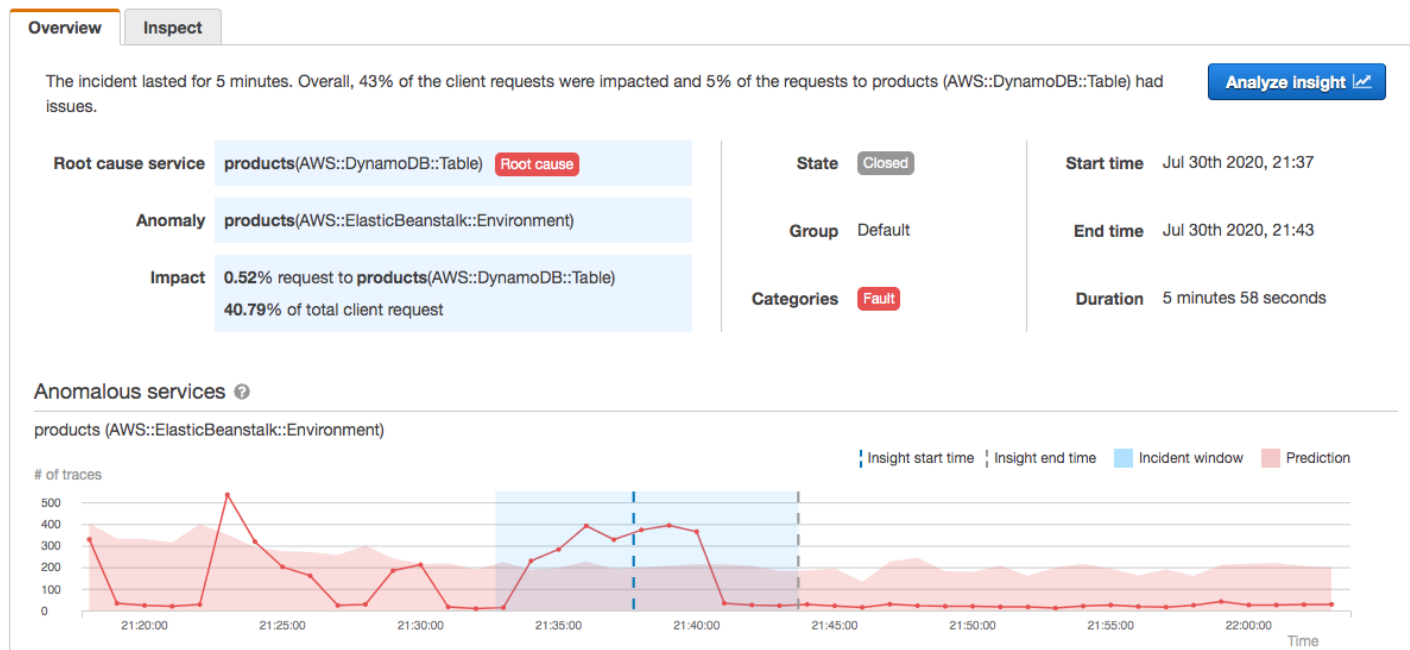
- Quel est le problème sous-jacent ?
- Quelle en est la cause première ?

- Quel en est l'impact ?

La section Services anormaux présente une chronologie pour chaque service qui illustre l'évolution du taux de défaillance au cours de l'incident. La chronologie indique le nombre de traces comportant des défaillances superposées sur une bande continue qui indique le nombre de défaillances attendu en fonction du volume de trafic enregistré. La durée de l'aperçu est visualisée par la fenêtre Incident. La fenêtre d'incident commence lorsque X-Ray observe que la métrique devient anormale et persiste tant que l'aperçu est actif.

L'exemple suivant montre une augmentation du nombre de défaillances à l'origine d'un incident :

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

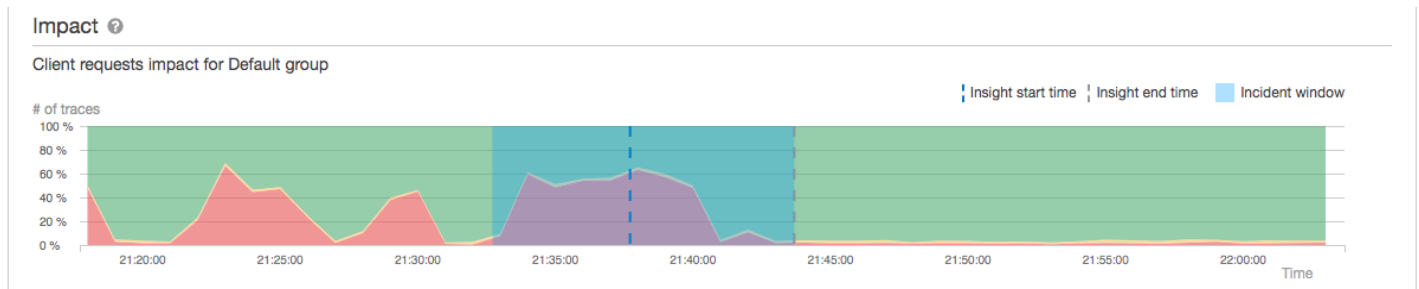


La section Cause première affiche une carte de trace axée sur le service responsable et le chemin concerné. Vous pouvez masquer les nœuds non affectés en sélectionnant l'icône en forme d'œil en haut à droite de la carte des causes profondes. Le service responsable est le nœud le plus en aval où X-Ray a identifié une anomalie. Il peut s'agir d'un service que vous avez instrumenté ou d'un service externe que votre service a appelé auprès d'un client instrumenté. Par exemple, si vous appelez Amazon DynamoDB avec un client SDK AWS instrumenté, l'augmentation du nombre de défaillances provenant de DynamoDB permet d'obtenir un aperçu dont DynamoDB est la cause première.

Pour étudier plus en détail la cause première, sélectionnez Afficher les détails de la cause première sur le graphique des causes premières. Vous pouvez utiliser la page Analytics pour rechercher la cause première et les messages associés. Pour de plus amples informations, veuillez consulter [Interaction avec la console Analytics](#).



Les défaillances qui se poursuivent en amont de la carte peuvent avoir un impact sur plusieurs nœuds et provoquer de multiples anomalies. Si une erreur est retransmise à l'utilisateur qui a fait la demande, il en résulte une erreur du client. Il s'agit d'un défaut dans le nœud racine de la carte de trace. Le graphique d'impact fournit une chronologie de l'expérience client pour l'ensemble du groupe. Cette expérience est calculée en fonction des pourcentages des états suivants : Fault, Error, Throttle et OK.



Cet exemple montre une augmentation du nombre de traces associées à une défaillance au niveau du nœud racine au moment d'un incident. Les incidents dans les services en aval ne correspondent pas toujours à une augmentation des erreurs des clients.

Choisissez **Analyze insight** pour ouvrir la console X-Ray Analytics dans une fenêtre dans laquelle vous pouvez examiner en détail l'ensemble des traces à l'origine de l'aperçu. Pour de plus amples informations, veuillez consulter [Interaction avec la console Analytics](#).

Comprendre l'impact

AWS X-Ray mesure l'impact causé par un problème récurrent dans le cadre de la génération d'informations et de notifications. L'impact est mesuré de deux manières :

- Impact sur le **groupe** X-Ray

- Impact sur le service responsable

Cet impact est déterminé par le pourcentage de demandes qui échouent ou sont à l'origine d'une erreur au cours d'une période donnée. Cette analyse d'impact vous permet de déterminer la gravité et la priorité du problème en fonction de votre scénario particulier. Cet impact est disponible dans le cadre de l'expérience de console, en plus des notifications d'informations.

Déduplication

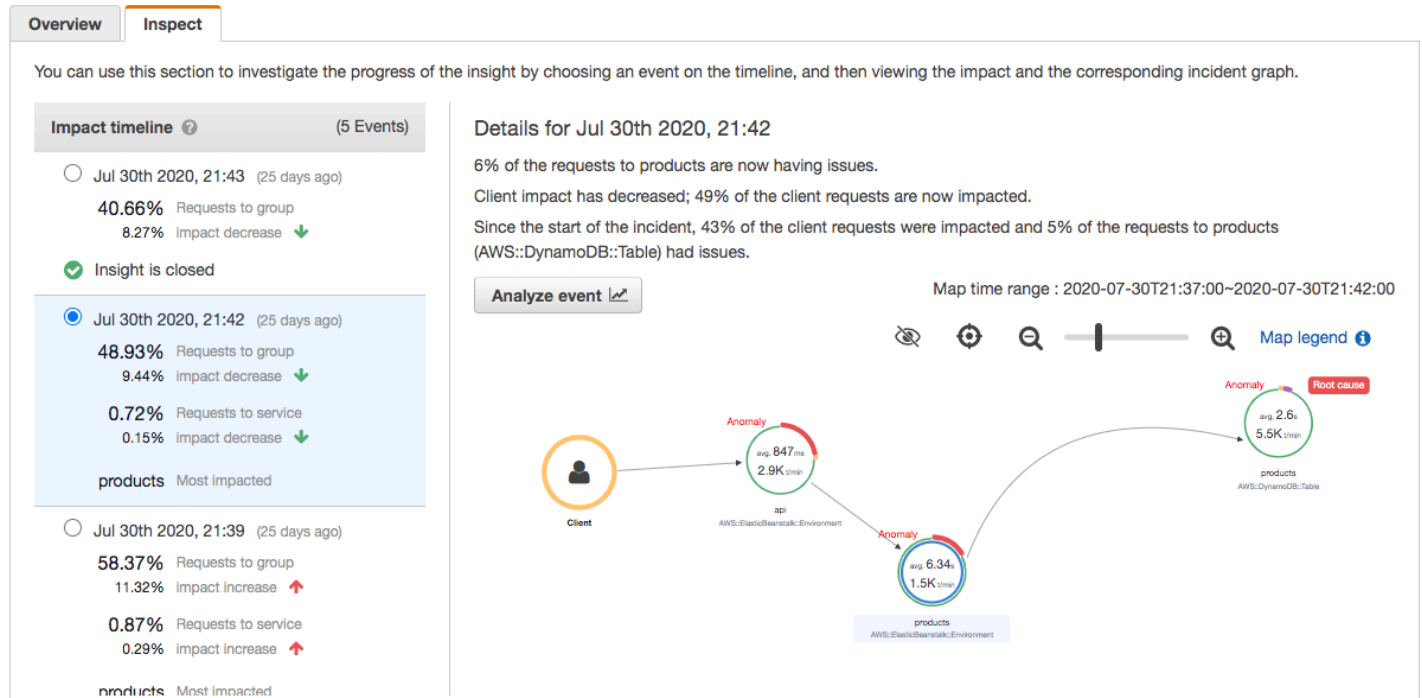
AWS X-Ray Insights déduplique les problèmes sur plusieurs microservices. Il utilise la détection des anomalies pour déterminer le service à l'origine d'un problème, déterminer si d'autres services connexes présentent un comportement anormal dû à la même cause première et enregistre le résultat sous la forme d'un aperçu unique.

Passez en revue la progression d'un aperçu

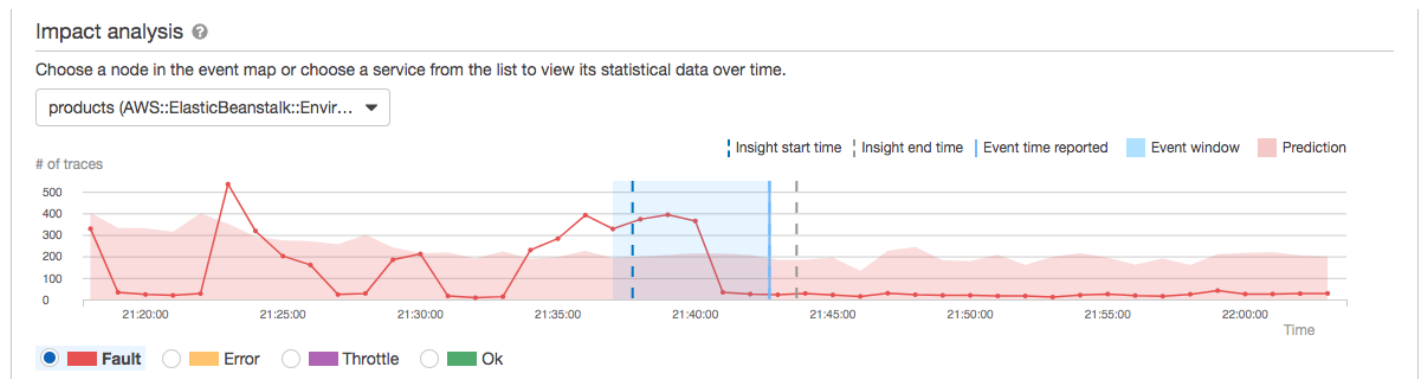
X-Ray réévalue régulièrement les informations jusqu'à ce qu'elles soient résolues, et enregistre chaque modification intermédiaire notable sous forme de [notification](#), qui peut être envoyée sous forme d'événement Amazon EventBridge . Cela vous permet de créer des processus et des flux de travail pour déterminer l'évolution du problème au fil du temps et de prendre les mesures appropriées, telles que l'envoi d'un e-mail ou l'intégration à un système d'alerte utilisant EventBridge.

Vous pouvez consulter les événements de l'incident dans la chronologie de l'impact sur la page Inspect. Par défaut, la chronologie affiche le service le plus touché jusqu'à ce que vous choisissiez un autre service.

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



Pour consulter une carte de suivi et des graphiques d'un événement, sélectionnez-le dans la chronologie de l'impact. La carte de suivi indique les services de votre application concernés par l'incident. Sous Analyse d'impact, les graphiques indiquent la chronologie des défaillances pour le nœud sélectionné et pour les clients du groupe.



Pour examiner de manière plus approfondie les traces impliquées dans un incident, choisissez Analyser l'événement sur la page Inspecter. Vous pouvez utiliser la page Analytics pour affiner la liste des traces et identifier les utilisateurs concernés. Pour de plus amples informations, veuillez consulter [Interaction avec la console Analytics](#).

Interaction avec la console Analytics

La console AWS X-Ray Analytics est un outil interactif permettant d'interpréter les données de trace afin de comprendre rapidement les performances de votre application et de ses services sous-jacents. La console vous permet d'explorer, d'analyser et de visualiser les suivis grâce à des graphiques interactifs sur les temps de réponse et la chronologie.

Lorsque vous sélectionnez un élément dans la console Analytics, celle-ci crée des filtres afin d'afficher le sous-ensemble de suivis sélectionné. Vous pouvez affiner l'ensemble de données actif en appliquant des filtres de plus en plus précis. Pour ce faire, cliquez sur les graphiques, les panneaux de métriques et les champs associés à l'ensemble de suivis sélectionné.

Rubriques

- [Fonctions de la console](#)
- [Distribution des temps de réponse](#)
- [Activités chronologiques](#)
- [Exemples de flux de travail](#)
- [Observez les erreurs sur le graphique des services](#)
- [Identifier les pics des temps de réponse](#)
- [Afficher tous les suivis marqués d'un code d'état](#)
- [Affichage de l'ensemble des éléments d'un sous-groupe et associés à un utilisateur](#)
- [Comparaison de deux ensembles de suivis avec des critères différents](#)
- [Identifier un suivi qui vous intéresse et afficher ses détails](#)

Fonctions de la console

La console X-Ray Analytics utilise les fonctionnalités clés suivantes pour regrouper, filtrer, comparer et quantifier les données de suivi.

Fonctionnalités

Fonctionnalité	Description
Groups (Groupes)	Le premier groupe sélectionné est Default. Pour modifier le groupe extrait, sélectionnez un groupe différent dans le menu situé à droite

Fonctionnalité	Description
	de la barre de recherche principale pour les expressions de filtre. Pour plus d'informations sur les groupes, consultez Utilisation d'expressions de filtrage avec les groupes.
Retrieved traces (Suivis extraits)	Par défaut, la console Analytics génère des graphiques basés sur tous les suivis du groupe sélectionné. Les suivis extraits représentent tous les suivis de l'ensemble sur lequel vous travaillez. Le nombre de suivis est affiché dans cette vignette. Les expressions de filtre que vous appliquez à la barre de recherche principale affinent et mettent à jour les suivis extraits.
Show in charts/Hide from charts (Afficher dans les graphiques/Masquer dans les graphiques)	Une fonctionnalité de basculement pour comparer le groupe actif aux suivis extraits. Pour comparer les données du groupe aux filtres actifs, sélectionnez Show in charts (Afficher dans les graphiques). Pour supprimer cette vue des graphiques, sélectionnez Hide from charts (Masquer dans les graphiques).
Filtered trace set A (Ensemble de suivis filtrés A)	En interagissant avec les graphiques et les tableaux, appliquez des filtres pour créer les critères du jeu de traces filtré A. Au fur et à mesure que les filtres sont appliqués, le nombre de traces applicables et le pourcentage de traces extraites par rapport au total sont calculés dans cette vignette. Les filtres sont renseignés comme des balises au sein de la vignette Filtered trace set A (Ensemble de suivis filtrés A) et peuvent également être supprimés de la vignette.

Fonctionnalité	Description
Refine (Affiner)	Cette fonction met à jour l'ensemble des suivis extraits en fonction des filtres appliqués à l'ensemble de suivis A. L'affinage de l'ensemble permet d'actualiser le jeu de suivis extraits sur lequel vous travaillez en fonction des filtres de l'ensemble des suivis A. L'ensemble des suivis extraits sur lequel vous travaillez est un exemple de sous-ensemble de tous les suivis du groupe.
Filtered trace set B (Ensemble de suivis filtrés B)	Une fois créé, le jeu de traces filtré B est une copie du jeu de traces filtré A. Pour comparer les deux ensembles de traces, effectuez de nouvelles sélections de filtres qui s'appliqueront au jeu de traces B, tandis que le jeu de traces A reste fixe. À mesure que les filtres sont appliqués, le nombre de suivis applicables ainsi que le pourcentage de suivis par rapport au nombre total de suivis extraits sont recalculés au sein de la vignette. Les filtres sont renseignés comme des balises au sein de la vignette Filtered trace set B (Ensemble de suivis filtrés B). Ils peuvent également être supprimés de la vignette.

Fonctionnalité	Description
Response Time Root Cause Entity Paths (Chemins d'entité constituant la cause racine du temps de réponse)	Tableau des chemins d'entités enregistrés. X-Ray détermine la trajectoire de votre trace qui est la cause la plus probable du temps de réponse. Le format indique une hiérarchie d'entités qui a été rencontrée, qui constitue la cause racine du temps de réponse. Utilisez ces lignes pour filtrer les erreurs de temps de réponse récurrentes. Pour plus d'informations sur la personnalisation des filtres de cause racine et sur l'obtention des données via l'API, consultez Extraction et affinage de l'analyse de la cause racine .
Delta (◆)	Colonne qui est ajoutée aux tables de métriques lorsque les ensembles A et B sont tous les deux actifs. La colonne Delta calcule la différence de suivis entre l'ensemble A et l'ensemble B sous forme de pourcentage.

Distribution des temps de réponse

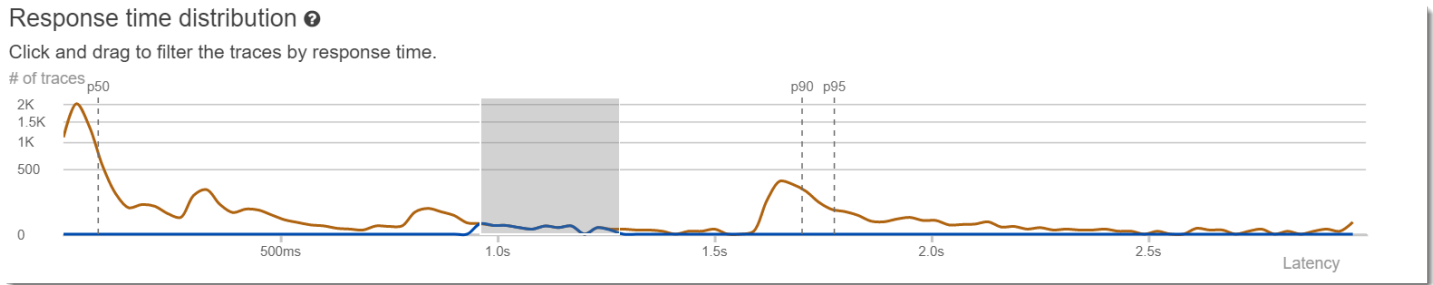
La console X-Ray Analytics génère deux graphiques principaux pour vous aider à visualiser les traces : la distribution du temps de réponse et l'activité des séries chronologiques. Cette section fournit des exemples de chacun de ces graphiques, et explique les principes de base pour savoir les lire.

Voici les couleurs associées au graphique linéaire des temps de réponse (le graphique chronologique utilise le même code couleurs) :

- Toutes les traces du groupe — gris
- Traces récupérées — orange
- Set de traces filtrées A — vert
- Set de traces filtrées B — bleu

Exemple — Répartition du temps de réponse

La distribution des temps de réponse est un graphique qui montre le nombre de suivis correspondant à un temps de réponse donné. Cliquez et faites glisser pour effectuer des sélections au sein du graphique de distribution des temps de réponse. Lorsque vous effectuez une sélection, un filtre est créé pour l'ensemble des suivis actifs. Cet ensemble est appelé `responseTime`, et correspond à tous les suivis au sein d'un temps de réponse spécifique.

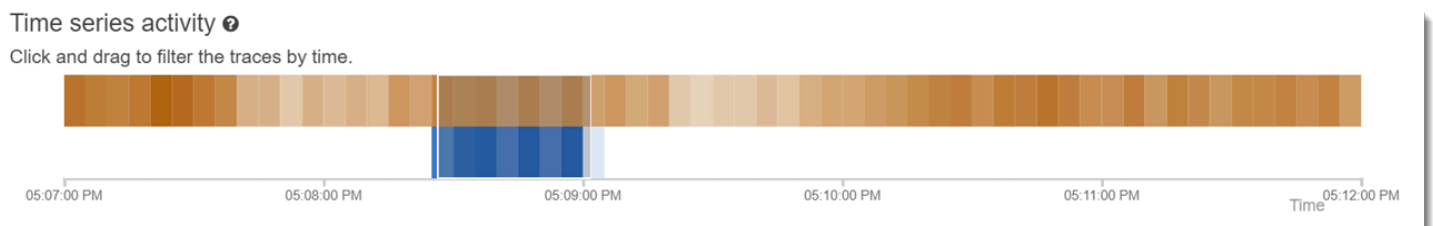


Activités chronologiques

Le graphique sur les activités chronologiques affiche le nombre de suivis sur une période donnée. Les indicateurs de couleurs reflètent le graphique linéaire de distribution des temps de réponse. Plus le bloc de couleur est foncé et rempli dans les activités chronologiques, plus la quantité de suivis représentée est élevée pour la période de temps donnée.

Exemple — Activité liée aux séries chronologiques

Cliquez et faites glisser pour effectuer des sélections au sein du graphique d'activités chronologiques. Lorsque vous effectuez une sélection, un filtre appelé `timeRange` est créé pour l'ensemble des suivis actifs au sein d'une période de temps spécifique.



Exemples de flux de travail

Les exemples suivants présentent des cas d'utilisation courants de la console X-Ray Analytics. Chaque exemple montre une fonctionnalité clé de la console. En tant que groupe, les exemples suivent un flux de travail basique pour la résolution de problèmes. Les étapes expliquent comment

repérer d'abord les nœuds défectueux, puis comment interagir avec la console Analytics pour générer automatiquement des requêtes comparatives. Une fois que vous avez réduit la portée par le biais de requêtes, vous examinez enfin les détails des suivis d'intérêt pour déterminer ce qui nuit à l'état de votre service.

Observez les erreurs sur le graphique des services

La carte de suivi indique l'état de santé de chaque nœud en le colorant en fonction du rapport entre les appels réussis et les erreurs et les défaillances. Lorsque votre nœud affiche un pourcentage rouge, cela indique une erreur. Utilisez la console X-Ray Analytics pour l'étudier.

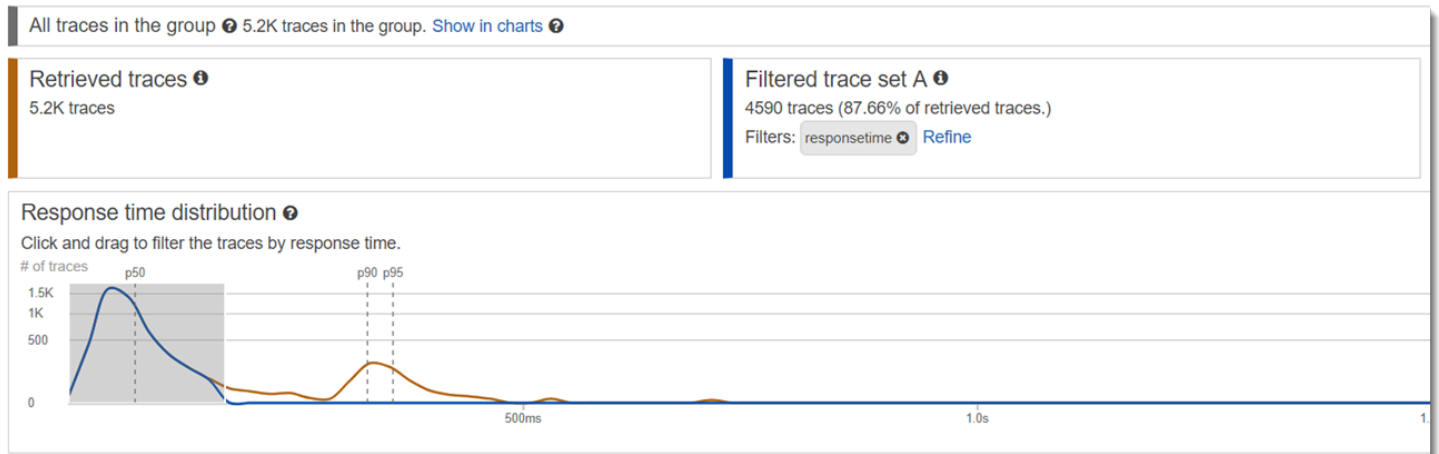
Pour plus d'informations sur la façon de lire la carte de trace, reportez-vous à la section [Affichage de la carte de trace](#).



Identifier les pics des temps de réponse

À l'aide du graphique sur la distribution des temps de réponse, vous pouvez observer les pics. Lorsque vous sélectionnez un pic dans les temps de réponse, les tables en dessous des graphiques se mettent à jour afin d'afficher toutes les métriques associées, comme les codes d'état.

Lorsque vous cliquez et faites glisser le pointeur, X-Ray sélectionne et crée un filtre. Il est grisé au-dessus des lignes du graphique. Vous pouvez alors faire glisser cet élément grisé vers la gauche ou vers la droite de la distribution pour modifier la sélection et filtrer.



Afficher tous les suivis marqués d'un code d'état

Vous pouvez afficher le détail des suivis au sein du pic sélectionné à l'aide des tables de métriques en dessous des graphiques. En cliquant sur une ligne de la table HTTP STATUS CODE (CODE D'ÉTAT HTTP), vous créez automatiquement un filtre pour l'ensemble de données actif. Par exemple, vous pouvez afficher tous les suivis du code d'état 500. Une balise de filtre est alors créée dans la vignette de l'ensemble des suivis nommé `http.status`.

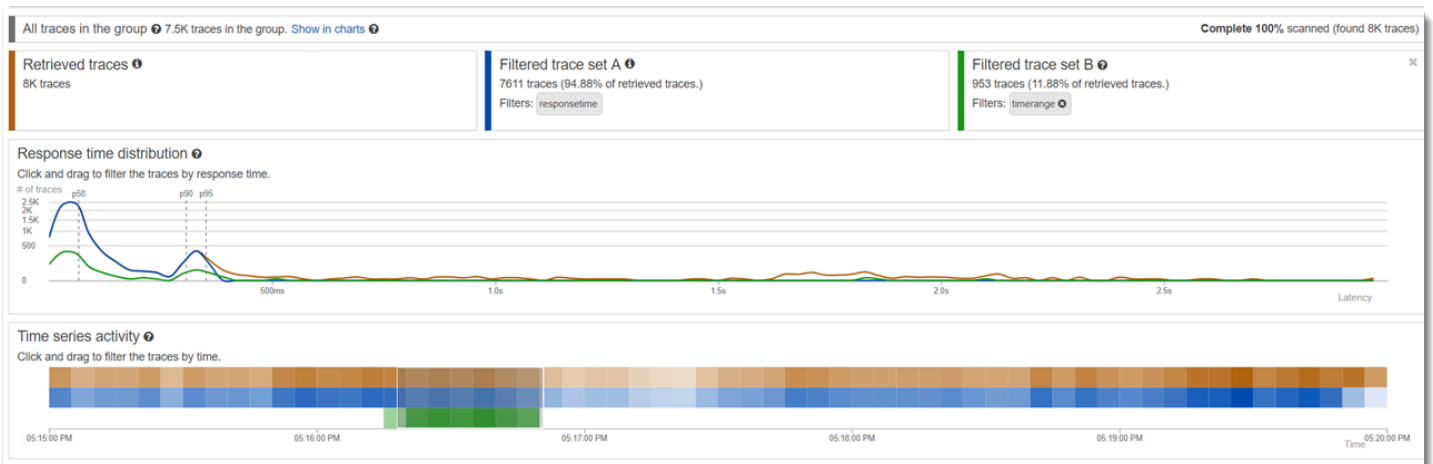
Affichage de l'ensemble des éléments d'un sous-groupe et associés à un utilisateur

Affichez le détail de l'ensemble des erreurs en fonction de l'utilisateur, de l'URL, de la cause racine du temps de réponse et d'autres attributs prédéfinis. Par exemple, pour filtrer l'ensemble des suivis dont le code d'état est 500, sélectionnez une ligne dans la table USERS (UTILISATEURS). Deux balises de filtre sont alors générées dans la vignette de l'ensemble des suivis : `http.status`, comme indiqué précédemment, et `user`.

Comparaison de deux ensembles de suivis avec des critères différents

Comparez plusieurs utilisateurs et leurs requêtes POST pour identifier des divergences et des corrélations. Appliquez votre premier ensemble de filtres. Ils sont indiqués par une ligne bleue dans le graphique de distribution des temps de réponse. Ensuite, sélectionnez Compare (Comparer). Pour commencer, une copie des filtres est créée sur l'ensemble des suivis A.

Ensuite, définissez un nouvel ensemble de filtres à appliquer à l'ensemble des suivis B. Ce deuxième ensemble est représenté par une ligne verte. L'exemple suivant montre les différentes courbes conformément au code de couleurs (bleu et vert).



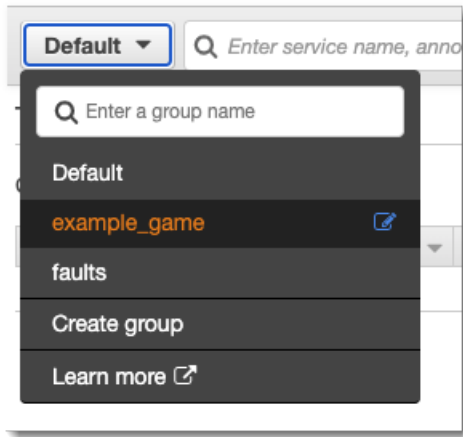
Identifier un suivi qui vous intéresse et afficher ses détails

À mesure que vous réduisez le champ d'application à l'aide des filtres de la console, la liste des suivis en dessous des tables de métriques devient plus parlante. La table de la liste des suivis combine des informations sur l'URL, l'UTILISATEUR, et le CODE D'ÉTAT dans une même vue. Pour obtenir plus d'informations, sélectionnez une ligne de la table afin d'ouvrir la page détaillée du suivi et d'examiner sa chronologie et ses données brutes.

Configuration des groupes

Les groupes constituent un ensemble de suivis qui sont définis par une expression de filtre. Vous pouvez utiliser des groupes pour générer des graphiques de service supplémentaires et fournir CloudWatch des statistiques Amazon. Vous pouvez utiliser la AWS X-Ray console ou l'API X-Ray pour créer et gérer des groupes pour vos services. Cette rubrique explique comment créer et gérer des groupes à l'aide de la console X-Ray. Pour plus d'informations sur la gestion des groupes à l'aide de l'API X-Ray, consultez [Groups](#).

Vous pouvez créer des groupes de traces pour les cartes de traçage, les traces ou les analyses. Lorsque vous créez un groupe, celui-ci devient disponible sous forme de filtre dans le menu déroulant du groupe sur les trois pages : Trace Map, Traces et Analytics.



Les groupes sont identifiés par leur nom ou un Amazon Resource Name (ARN), et contiennent une expression de filtre. Le service compare les suivis entrants à l'expression, et les stocke en conséquence. Pour plus d'informations sur la création d'une expression de filtre, consultez [Utilisation d'expressions de filtre](#).

Le fait de mettre à jour l'expression de filtrage d'un groupe n'a pas pour effet de modifier les données déjà enregistrées. La mise à jour s'applique uniquement aux suivis ultérieurs. Cela peut entraîner la fusion des expressions nouvelles et anciennes dans le graphique. Pour éviter cela, supprimez le groupe actuel et créez-en un nouveau.

Note

Les groupes sont facturés en fonction du nombre de suivis récupérés qui correspondent à l'expression de filtrage. Pour en savoir plus, consultez [Pricing AWS X-Ray](#) (Tarification).

Rubriques

- [Créer un groupe](#)
- [Appliquer un groupe](#)
- [Modifier un groupe](#)
- [Cloner un groupe](#)
- [Supprimer un groupe](#)
- [Afficher les statistiques du groupe sur Amazon CloudWatch](#)

Créez un groupe

Note

Vous pouvez désormais configurer les groupes X-Ray depuis la CloudWatch console Amazon. Vous pouvez également continuer à utiliser la console X-Ray.

CloudWatch console

1. Connectez-vous à la CloudWatch console AWS Management Console et ouvrez-la à l'adresse <https://console.aws.amazon.com/cloudwatch/>.
2. Choisissez Paramètres dans le volet de navigation de gauche.
3. Choisissez Afficher les paramètres sous Groupes dans la section X-Ray Traces.
4. Choisissez Créer un groupe au-dessus de la liste des groupes.
5. Sur la page Créer un groupe, entrez le nom du groupe. Le nom d'un groupe peut comporter un maximum de 32 caractères, dont des caractères alphanumériques et des tirets. Les noms de groupes distinguent les majuscules et minuscules.
6. Entrez une expression de filtre. Pour plus d'informations sur la création d'une expression de filtre, consultez [Utilisation d'expressions de filtre](#). Dans l'exemple suivant, le groupe filtre les traces d'erreur provenant du service `api.example.com` et les demandes adressées au service dont le temps de réponse était supérieur ou égal à cinq secondes.

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

7. Dans Insights, activez ou désactivez l'accès aux insights pour le groupe. Pour en savoir plus sur les informations, consultez [Utilisation des informations issues de X-Ray](#).

☒ Enable insights

☐ Enable notifications

Deliver insight events using Amazon EventBridge.

8. Dans Balises, choisissez Ajouter une nouvelle balise pour saisir une clé de balise et, éventuellement, une valeur de balise. Continuez à ajouter des balises supplémentaires si vous le souhaitez. Les clés de tag doivent être uniques. Pour supprimer un tag, choisissez

Supprimer en dessous de chaque tag. Pour en savoir plus sur les identifications, consultez [Étiquetage des règles et des groupes d'échantillonnage aux rayons X](#).

Key	Value - optional
Q Enter key	Q Enter value
Remove	

9. Choisissez Créer un groupe.

X-Ray console

1. Connectez-vous à la console X-Ray AWS Management Console et ouvrez-la <https://console.aws.amazon.com/xray/chez> vous.
2. Ouvrez la page Créer un groupe depuis la page Groupes du volet de navigation de gauche ou depuis le menu du groupe sur l'une des pages suivantes : Trace Map, Traces et Analytics.
3. Sur la page Créer un groupe, entrez le nom du groupe. Le nom d'un groupe peut comporter un maximum de 32 caractères, dont des caractères alphanumériques et des tirets. Les noms de groupes distinguent les majuscules et minuscules.
4. Entrez une expression de filtre. Pour plus d'informations sur la création d'une expression de filtre, consultez [Utilisation d'expressions de filtre](#). Dans l'exemple suivant, le groupe filtre les traces d'erreur provenant du service `api.example.com` et les demandes adressées au service dont le temps de réponse était supérieur ou égal à cinq secondes.

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

5. Dans Insights, activez ou désactivez l'accès aux insights pour le groupe. Pour en savoir plus sur les informations, consultez [Utilisation des informations issues de X-Ray](#).

Enable Insights ☒

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

6. Dans Balises, entrez une clé de balise et, éventuellement, une valeur de balise. Lorsque vous ajoutez un tag, une nouvelle ligne apparaît pour vous permettre de saisir un autre tag. Les clés de tag doivent être uniques. Pour supprimer un tag, choisissez X à la fin de la ligne du tag. Pour en savoir plus sur les identifications, consultez [Étiquetage des règles et des groupes d'échantillonnage aux rayons X](#).

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

7. Choisissez Créer un groupe.

Appliquer un groupe

CloudWatch console

1. Connectez-vous à la CloudWatch console AWS Management Console et ouvrez-la à l'adresse <https://console.aws.amazon.com/cloudwatch/>.
2. Ouvrez l'une des pages suivantes depuis le volet de navigation sous X-Ray Traces :
 - Carte de suivi
 - Suivis
3. Entrez un nom de groupe dans le filtre de groupe Filter by X-Ray. Les données affichées sur la page changent pour correspondre à l'expression de filtre définie dans le groupe.

X-Ray console

1. Connectez-vous à la console X-Ray AWS Management Console et ouvrez-la <https://console.aws.amazon.com/xray/chez> vous.
2. Ouvrez l'une des pages suivantes depuis le volet de navigation :
 - Carte de suivi
 - Suivis
 - Analyse
3. Dans le menu du groupe, choisissez le groupe dans lequel vous avez créé le groupe [the section called “Créez un groupe”](#). Les données affichées sur la page changent pour correspondre à l'expression de filtre définie dans le groupe.

Modifier un groupe

CloudWatch console

1. Connectez-vous à la CloudWatch console AWS Management Console et ouvrez-la à l'adresse <https://console.aws.amazon.com/cloudwatch/>.
2. Choisissez Paramètres dans le volet de navigation de gauche.
3. Choisissez Afficher les paramètres sous Groupes dans la section X-Ray Traces.
4. Choisissez un groupe dans la section Groupes, puis sélectionnez Modifier.
5. Bien que vous ne puissiez pas renommer un groupe, vous pouvez mettre à jour l'expression du filtre. Pour plus d'informations sur la création d'une expression de filtre, consultez [Utilisation d'expressions de filtre](#). Dans l'exemple suivant, le groupe filtre les traces d'erreur provenant du service `api.example.com`, où figure l'adresse URL de la demande `example/game`, et le temps de réponse des demandes était supérieur ou égal à cinq secondes.

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

6. Dans Insights, activez ou désactivez l'accès aux insights pour le groupe. Pour en savoir plus sur les informations, consultez [Utilisation des informations issues de X-Ray](#).

☒ Enable insights

☐ Enable notifications

Deliver insight events using Amazon EventBridge.

7. Dans Balises, choisissez Ajouter une nouvelle balise pour saisir une clé de balise et, éventuellement, une valeur de balise. Continuez à ajouter des balises supplémentaires si vous le souhaitez. Les clés de tag doivent être uniques. Pour supprimer un tag, choisissez Supprimer en dessous de chaque tag. Pour en savoir plus sur les identifications, consultez [Étiquetage des règles et des groupes d'échantillonnage aux rayons X](#).

Key

Value - optional

🔍 Enter key

🔍 Enter value

Remove

8. Lorsque vous avez terminé de mettre à jour le groupe, choisissez Mettre à jour le groupe.

X-Ray console

1. Connectez-vous à la console X-Ray AWS Management Console et ouvrez-la <https://console.aws.amazon.com/xray/chez> vous.
2. Procédez de l'une des manières suivantes pour ouvrir la page Modifier le groupe.
 - a. Sur la page Groupes, choisissez le nom d'un groupe pour le modifier.
 - b. Dans le menu des groupes de l'une des pages suivantes, pointez sur un groupe, puis choisissez Modifier.
 - Carte de suivi
 - Suivis
 - Analyse
3. Bien que vous ne puissiez pas renommer un groupe, vous pouvez mettre à jour l'expression du filtre. Pour plus d'informations sur la création d'une expression de filtre, consultez [Utilisation d'expressions de filtre](#). Dans l'exemple suivant, le groupe filtre les traces d'erreur provenant du service `api.example.com`, où figure l'adresse URL de la demande `example/game`, et le temps de réponse des demandes était supérieur ou égal à cinq secondes.

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

4. Dans Insights, activez ou désactivez les informations et les notifications d'informations pour le groupe. Pour en savoir plus sur les informations, consultez [Utilisation des informations issues de X-Ray](#).

Enable Insights ☒

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

5. Dans Balises, modifiez les clés et les valeurs des balises. Les clés de tag doivent être uniques. Les valeurs des balises sont facultatives ; vous pouvez supprimer des valeurs si vous le souhaitez. Pour supprimer un tag, choisissez X à la fin de la ligne du tag. Pour en savoir plus sur les identifications, consultez [Étiquetage des règles et des groupes d'échantillonnage aux rayons X](#).

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

6. Lorsque vous avez terminé de mettre à jour le groupe, choisissez Mettre à jour le groupe.

Cloner un groupe

Le clonage d'un groupe crée un nouveau groupe qui possède l'expression de filtre et les balises d'un groupe existant. Lorsque vous clonez un groupe, le nouveau groupe porte le même nom que le groupe à partir duquel il a été cloné, avec un `-clone` ajout au nom.

CloudWatch console

1. Connectez-vous à la CloudWatch console AWS Management Console et ouvrez-la à l'adresse <https://console.aws.amazon.com/cloudwatch/>.
2. Choisissez Paramètres dans le volet de navigation de gauche.
3. Choisissez Afficher les paramètres sous Groupes dans la section X-Ray Traces.
4. Choisissez un groupe dans la section Groupes, puis cliquez sur Cloner.
5. Sur la page Créer un groupe, le nom du groupe est `group-name-clone`. Entrez éventuellement un nouveau nom pour le groupe. Le nom d'un groupe peut comporter un maximum de 32 caractères, dont des caractères alphanumériques et des tirets. Les noms de groupes distinguent les majuscules et minuscules.
6. Vous pouvez conserver l'expression de filtre du groupe existant ou éventuellement entrer une nouvelle expression de filtre. Pour plus d'informations sur la création d'une expression de filtre, consultez [Utilisation d'expressions de filtre](#). Dans l'exemple suivant, le groupe filtre les traces d'erreur provenant du service `api.example.com` et les demandes adressées au service dont le temps de réponse était supérieur ou égal à cinq secondes.

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

7. Dans Balises, modifiez les clés et les valeurs des balises, si nécessaire. Les clés de tag doivent être uniques. Les valeurs des balises sont facultatives ; vous pouvez supprimer des valeurs si vous le souhaitez. Pour supprimer un tag, choisissez X à la fin de la ligne du tag.

Pour en savoir plus sur les identifications, consultez [Étiquetage des règles et des groupes d'échantillonnage aux rayons X](#).

8. Choisissez Créer un groupe.

X-Ray console

1. Connectez-vous à la console X-Ray AWS Management Console et ouvrez-la <https://console.aws.amazon.com/xray/chez> vous.
2. Ouvrez la page Groupes dans le volet de navigation de gauche, puis choisissez le nom du groupe que vous souhaitez cloner.
3. Choisissez le groupe Cloner dans le menu Actions.
4. Sur la page Créer un groupe, le nom du groupe est *group-name*-clone. Entrez éventuellement un nouveau nom pour le groupe. Le nom d'un groupe peut comporter un maximum de 32 caractères, dont des caractères alphanumériques et des tirets. Les noms de groupes distinguent les majuscules et minuscules.
5. Vous pouvez conserver l'expression de filtre du groupe existant ou éventuellement entrer une nouvelle expression de filtre. Pour plus d'informations sur la création d'une expression de filtre, consultez [Utilisation d'expressions de filtre](#). Dans l'exemple suivant, le groupe filtre les traces d'erreur provenant du service `api.example.com` et les demandes adressées au service dont le temps de réponse était supérieur ou égal à cinq secondes.

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

6. Dans Balises, modifiez les clés et les valeurs des balises, si nécessaire. Les clés de tag doivent être uniques. Les valeurs des balises sont facultatives ; vous pouvez supprimer des valeurs si vous le souhaitez. Pour supprimer un tag, choisissez X à la fin de la ligne du tag. Pour en savoir plus sur les identifications, consultez [Étiquetage des règles et des groupes d'échantillonnage aux rayons X](#).
7. Choisissez Créer un groupe.

Supprimer un groupe

Suivez les étapes décrites dans cette section pour supprimer un groupe. Vous ne pouvez pas supprimer le groupe par défaut.

CloudWatch console

1. Connectez-vous à la CloudWatch console AWS Management Console et ouvrez-la à l'adresse <https://console.aws.amazon.com/cloudwatch/>.
2. Choisissez Paramètres dans le volet de navigation de gauche.
3. Choisissez Afficher les paramètres sous Groupes dans la section X-Ray Traces.
4. Choisissez un groupe dans la section Groupes, puis sélectionnez Supprimer.
5. Lorsque vous êtes invité à confirmer, choisissez Supprimer.

X-Ray console

1. Connectez-vous à la console X-Ray AWS Management Console et ouvrez-la <https://console.aws.amazon.com/xray/chez> vous.
2. Ouvrez la page Groupes dans le volet de navigation de gauche, puis choisissez le nom du groupe que vous souhaitez supprimer.
3. Dans le menu Actions, choisissez Supprimer le groupe.
4. Lorsque vous êtes invité à confirmer, choisissez Supprimer.

Afficher les statistiques du groupe sur Amazon CloudWatch

Après la création d'un groupe, les traces entrantes sont vérifiées par rapport à l'expression du filtre du groupe lorsqu'elles sont stockées dans le service X-Ray. Les statistiques relatives au nombre de traces correspondant à chaque critère sont publiées sur Amazon CloudWatch toutes les minutes. Choisir Afficher la métrique sur la page Modifier le groupe ouvre la CloudWatch console sur la page Métrique. Pour plus d'informations sur l'utilisation des CloudWatch métriques, consultez la section [Utilisation d'Amazon CloudWatch Metrics](#) dans le guide de CloudWatch l'utilisateur Amazon.

CloudWatch console

1. Connectez-vous à la CloudWatch console AWS Management Console et ouvrez-la à l'adresse <https://console.aws.amazon.com/cloudwatch/>.
2. Choisissez Paramètres dans le volet de navigation de gauche.
3. Choisissez Afficher les paramètres sous Groupes dans la section X-Ray Traces.
4. Choisissez un groupe dans la section Groupes, puis sélectionnez Modifier.
5. Sur la page Modifier le groupe, choisissez Afficher la métrique.

La page Metrics de la CloudWatch console s'ouvre dans un nouvel onglet.

X-Ray console

1. Connectez-vous à la console X-Ray AWS Management Console et ouvrez-la <https://console.aws.amazon.com/xray/chez> vous.
2. Ouvrez la page Groupes dans le volet de navigation de gauche, puis choisissez le nom du groupe dont vous souhaitez consulter les métriques.
3. Sur la page Modifier le groupe, choisissez Afficher la métrique.

La page Metrics de la CloudWatch console s'ouvre dans un nouvel onglet.

Configuration de règles d'échantillonnage

Vous pouvez utiliser la AWS X-Ray console pour configurer les règles d'échantillonnage pour vos services. Le SDK X-Ray et ceux Services AWS qui prennent en charge le [traçage actif](#) avec configuration d'échantillonnage utilisent des règles d'échantillonnage pour déterminer les demandes à enregistrer.

Rubriques

- [Configuration de règles d'échantillonnage](#)
- [Personnalisation des règles d'échantillonnage](#)
- [Options de règle d'échantillonnage](#)
- [Exemples de règles d'échantillonnage](#)
- [Configuration de votre service afin d'utiliser les règles d'échantillonnage](#)
- [Affichage des résultats d'échantillonnage](#)
- [Étapes suivantes](#)

Configuration de règles d'échantillonnage

Vous pouvez configurer l'échantillonnage pour les cas d'utilisation suivants :

- API Gateway Entrypoint : API Gateway prend en charge l'échantillonnage et le suivi actif. Pour activer le suivi actif sur une étape d'API, consultez [Support de suivi actif d'Amazon API Gateway pour AWS X-Ray](#).

- AWS AppSync— AWS AppSync prend en charge l'échantillonnage et le traçage actif. Pour activer le suivi actif des AWS AppSync demandes, voir [Tracing with AWS X-Ray](#).
- SDK Instrument X-Ray sur des plateformes informatiques : lorsque vous utilisez des plateformes de calcul telles qu'Amazon EC2, Amazon ECS ou AWS Elastic Beanstalk, l'échantillonnage est pris en charge lorsque l'application a été instrumentée avec le dernier SDK X-Ray.

Personnalisation des règles d'échantillonnage

En personnalisant les règles d'échantillonnage, vous pouvez contrôler la quantité de données que vous enregistrez. Vous pouvez également modifier le comportement d'échantillonnage sans modifier ni redéployer votre code. Les règles d'échantillonnage indiquent au SDK X-Ray le nombre de demandes à enregistrer pour un ensemble de critères. Par défaut, le SDK X-Ray enregistre la première demande chaque seconde, et 5 % des demandes supplémentaires. Une demande par seconde est le réservoir. Ceci garantit qu'au moins une trace est enregistrée chaque seconde aussi longtemps que le service traite les demandes. 5 % est la fréquence à laquelle les demandes supplémentaires au-delà du réservoir sont échantillonnées.

Vous pouvez configurer le SDK X-Ray pour lire les règles d'échantillonnage à partir d'un document JSON que vous incluez dans votre code. Cependant, lorsque vous exécutez plusieurs instances de votre service, chaque instance exécute l'échantillonnage de manière indépendante. Ceci entraîne une augmentation du pourcentage global des demandes échantillonnées, car les réservoirs de toutes les instances sont effectivement additionnés. En outre, pour mettre à jour les règles d'échantillonnage locales, vous devez redéployer votre code.

En définissant des règles d'échantillonnage dans la console X-Ray et en [configurant le SDK](#) pour lire les règles du service X-Ray, vous pouvez éviter ces deux problèmes. Le service gère le réservoir pour chaque règle et attribue les quotas à chaque instance de votre service pour distribuer le réservoir de façon uniforme, en fonction du nombre d'instances qui sont en cours d'exécution. La limite de réservoir est calculée selon les règles que vous avez définies. Les règles étant configurées dans le service, vous pouvez les gérer sans effectuer de déploiements supplémentaires.

Note

X-Ray applique les règles d'échantillonnage dans la mesure du possible et, dans certains cas, le taux d'échantillonnage effectif peut ne pas correspondre exactement aux règles d'échantillonnage configurées. Cependant, au fil du temps, le nombre de demandes échantillonnées devrait être proche du pourcentage configuré.

Vous pouvez désormais configurer les règles d'échantillonnage X-Ray depuis la CloudWatch console Amazon. Vous pouvez également continuer à utiliser la console X-Ray.

CloudWatch console

Pour configurer les règles d'échantillonnage dans la CloudWatch console

1. Connectez-vous à la CloudWatch console AWS Management Console et ouvrez-la à l'adresse <https://console.aws.amazon.com/cloudwatch/>.
2. Choisissez Paramètres dans le volet de navigation de gauche.
3. Choisissez Afficher les paramètres sous Règles d'échantillonnage dans la section X-Ray Traces.
4. Pour créer une règle, choisissez Create sampling rule (Créer une règle d'échantillonnage).

Pour modifier une règle, choisissez-en une, puis cliquez sur Modifier pour la modifier.

Pour supprimer une règle, choisissez-en une, puis cliquez sur Supprimer pour la supprimer.

X-Ray console

Pour configurer les règles d'échantillonnage dans la console X-Ray

1. Ouvrez la [console X-Ray](#).
2. Choisissez Sampling dans le volet de navigation de gauche.
3. Pour créer une règle, choisissez Create sampling rule (Créer une règle d'échantillonnage).

Pour modifier une règle, choisissez le nom d'une règle.

Pour supprimer une règle, choisissez une règle et utilisez le menu Actions pour la supprimer.

Options de règle d'échantillonnage

Les options suivantes sont disponibles pour chaque règle. Les valeurs de chaîne peuvent comporter des caractères génériques correspondant à un seul caractère (?) ou à zéro caractère ou plus (*).

Options de règle d'échantillonnage

- Nom de la règle (chaîne) : nom unique de la règle.

- **Priorité** (entier compris entre 1 et 9999) : priorité de la règle d'échantillonnage. Les services évaluent les règles dans l'ordre croissant de priorité, et prennent une décision d'échantillonnage avec la première règle correspondante.
- **Réservoir** (entier non négatif) — Nombre fixe de demandes correspondantes à l'instrument par seconde, avant l'application du taux fixe. Le réservoir n'est pas utilisé directement par les services, mais s'applique à tous les services qui utilisent la règle collectivement.
- **Taux** (entier compris entre 0 et 100) : pourcentage de demandes correspondantes à l'instrument, une fois le réservoir épuisé. Lorsque vous configurez une règle d'échantillonnage dans la console, choisissez un pourcentage compris entre 0 et 100. Lorsque vous configurez une règle d'échantillonnage dans un SDK client à l'aide d'un document JSON, fournissez une valeur en pourcentage comprise entre 0 et 1.
- **Nom du service (chaîne)** : nom du service instrumenté, tel qu'il apparaît sur la carte de suivi.
 - X-Ray SDK : nom du service que vous configurez sur l'enregistreur.
 - Amazon API Gateway — *api-name/stage*.
- **Type de service (chaîne)** : type de service tel qu'il apparaît sur la carte de suivi. Pour le SDK X-Ray, définissez le type de service en appliquant le plug-in approprié :
 - `AWS::ElasticBeanstalk::Environment` — Un AWS Elastic Beanstalk environnement (plugin).
 - `AWS::EC2::Instance` — Une EC2 instance Amazon (plugin).
 - `AWS::ECS::Container` — Un conteneur Amazon ECS (plugin).
 - `AWS::APIGateway::Stage` — Une étape Amazon API Gateway.
 - `AWS::AppSync::GraphQLAPI` — Une demande AWS AppSync d'API.
- **Host (chaîne)** — Le nom d'hôte indiqué dans l'en-tête de l'hôte HTTP.
- **Méthode HTTP (chaîne)** — Méthode de la requête HTTP.
- **Chemin de l'URL (chaîne)** — Le chemin URL de la demande.
 - X-Ray SDK : partie du chemin de l'URL de la requête HTTP.
- **ARN de la ressource (chaîne)** — L'ARN de la AWS ressource exécutant le service.
 - X-Ray SDK — Non pris en charge. Le kit SDK peut uniquement utiliser des règles avec les ressources ARN définies sur `*`.
 - Amazon API Gateway — L'ARN du stage.
- **(Facultatif) Attributs (clé et valeur)** — Attributs de segment connus lorsque la décision d'échantillonnage est prise.

- X-Ray SDK — Non pris en charge. Le kit SDK ignore les règles qui spécifient les attributs.
- Amazon API Gateway — En-têtes issus de la requête HTTP d'origine.

Exemples de règles d'échantillonnage

Exemple — Règle par défaut sans réservoir et faible taux

Vous pouvez modifier le réservoir et la fréquence de la règle par défaut. Par défaut, la règle s'applique à toutes les demandes qui ne correspondent à aucune autre règle.

- Réservoir : **0**
- Taux : **5 (0.05**si configuré à l'aide d'un document JSON)

Exemple — Règle de débogage pour suivre toutes les demandes relatives à un itinéraire problématique

Une règle prioritaire est appliquée temporairement pour le débogage.

- Nom de la règle : **DEBUG - history updates**
- Priorité : **1**
- Réservoir : **1**
- Taux : **100 (1**si configuré à l'aide d'un document JSON)
- Nom du service : **Scorekeep**
- Type de service : *****
- Hôte : *****
- Méthode HTTP : **PUT**
- Chemin de l'URL : **/history/***
- ARN de la ressource : *****

Exemple — Taux minimum plus élevé pour POSTs

- Nom de la règle : **POST minimum**
- Priorité : **100**
- Réservoir : **10**

- Taux : **10** (.1 si configuré à l'aide d'un document JSON)
- Nom du service : *
- Type de service : *
- Hôte : *
- Méthode HTTP : **POST**
- Chemin de l'URL : *
- ARN de la ressource : *

Configuration de votre service afin d'utiliser les règles d'échantillonnage

Le SDK X-Ray nécessite une configuration supplémentaire pour utiliser les règles d'échantillonnage que vous configurez dans la console. Consultez la rubrique de configuration de votre langage pour en savoir plus sur la configuration d'une stratégie d'échantillonnage :

- Java: [Règles d'échantillonnage](#)
- Allez : [Règles d'échantillonnage](#)
- Node.js : [Règles d'échantillonnage](#)
- Python: [Règles d'échantillonnage](#)
- Rubis : [Règles d'échantillonnage](#)
- .NET : [Règles d'échantillonnage](#)

Pour API Gateway, consultez [Support de suivi actif d'Amazon API Gateway pour AWS X-Ray](#).

Affichage des résultats d'échantillonnage

La page d'échantillonnage de la console X-Ray affiche des informations détaillées sur la manière dont vos services utilisent chaque règle d'échantillonnage.

La colonne Trend (Tendance) montre comment la règle a été utilisée au cours des dernières minutes. Chaque colonne indique les statistiques pour une fenêtre de 10 secondes.

Échantillonnage des statistiques

- Règle correspondante totale : nombre de demandes correspondant à cette règle. Ce nombre n'inclut pas les demandes qui pourraient avoir une correspondance avec cette règle, mais celles qui ont en premier une règle de priorité.

- Total échantillonné : nombre de demandes enregistrées.
- Échantillonné avec un taux fixe : nombre de demandes échantillonnées en appliquant le taux fixe de la règle.
- Échantillonné avec limite de réservoir : nombre de demandes échantillonnées à l'aide d'un quota attribué par X-Ray.
- Emprunté au réservoir : nombre de demandes échantillonnées en empruntant auprès du réservoir. La première fois qu'un service fait correspondre une demande à une règle, aucun quota ne lui a encore été attribué par X-Ray. Toutefois, si le réservoir est d'au moins 1, le service emprunte une trace par seconde jusqu'à ce que X-Ray attribue un quota.

Pour plus d'informations sur l'échantillonnage des statistiques et la façon dont les services utilisent les règles d'échantillonnage, consultez [Utilisation de règles d'échantillonnage avec l'API X-Ray](#).

Étapes suivantes

Vous pouvez utiliser l'API X-Ray pour gérer les règles d'échantillonnage. Avec l'API, vous pouvez créer et mettre à jour les règles par programmation ou en fonction d'une planification, en réponse à des alarmes ou à des notifications. Consultez [Configuration de l'échantillonnage, des groupes et des paramètres de chiffrement avec l'API AWS X-Ray](#) pour obtenir des instructions et des exemples de règles supplémentaires.

Le SDK X-Ray peut Services AWS également utiliser l'API X-Ray pour lire les règles d'échantillonnage, communiquer les résultats d'échantillonnage et obtenir des cibles d'échantillonnage. Les services doivent suivre la fréquence à laquelle ils appliquent chaque règle, évaluer les règles en fonction de leur priorité et emprunter auprès du réservoir lorsqu'une demande correspond à une règle pour laquelle X-Ray n'a pas encore attribué de quota au service. Pour plus de détails sur la façon dont un service utilise l'API pour l'échantillonnage, reportez-vous à la section [Utilisation de règles d'échantillonnage avec l'API X-Ray](#).

Lorsque le SDK X-Ray appelle Sampling APIs, il utilise le daemon X-Ray comme proxy. Si vous utilisez déjà le port TCP 2000, vous pouvez configurer le démon pour exécuter le proxy sur un autre port. Consultez [Configuration du AWS X-Ray daemon](#) pour plus de détails.

Liens profonds sur console

Vous pouvez utiliser des itinéraires et des requêtes pour créer des liens profonds vers des traces spécifiques, ou des vues filtrées des traces et de la carte des traces.

Pages de console

- Page d'accueil — [xray/home#/welcome](#)
- Commencer [xray/home#/getting](#) — démarrer
- Tracer la carte — [xray/home#/service-map](#)
- Des traces — [xray/home#/traces](#)

Suivis

Vous pouvez générer des liens pour des vues chronologiques, brutes et de cartographie de suivis individuels.

Tracez la chronologie — [xray/home#/traces/*trace-id*](#)

Données de trace brutes — [xray/home#/traces/*trace-id*/raw](#)

Exemple — données de trace brutes

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

Expressions de filtrage

Lien vers une liste filtrée de suivis.

Affichage des traces filtrées — [xray/home#/traces?filter=*filter-expression*](#)

Exemple — expression de filtre

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

Exemple — expression de filtre (encodée en URL)

```
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
```

Pour plus d'informations sur les expressions de filtrage, consultez [Utilisation d'expressions de filtre](#).

Plage horaire

Spécifiez une durée ou une heure de début et de fin au format ISO86 01. Les plages horaires sont exprimées en UTC et peuvent durer jusqu'à 6 heures.

Durée — xray/home#/page?timeRange=*range-in-minutes*

Exemple — carte de trace de la dernière heure

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

Heure de début et de fin — xray/home#/page?timeRange=*start~end*

Exemple — plage de temps précise en secondes

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

Exemple — plage de temps précise en minutes

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

Région

Spécifiez un lien Région AWS vers les pages de cette région. Si vous n'indiquez pas de région, la console vous redirige vers la dernière région visitée.

Région — xray/home?region=*region*#/page

Exemple — carte de traçage dans l'ouest des États-Unis (Oregon) (us-west-2)

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

Lorsque vous incluez une région avec d'autres paramètres de requête, la requête de région précède le hachage et les X-Ray-specific requêtes suivent le nom de page.

Exemple — carte de suivi de la dernière heure dans l'ouest des États-Unis (Oregon) (us-west-2)

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

Combiné

Exemple — traces récentes avec filtre de durée

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

Sortie

- Page — Traces
- Plage de temps — 15 dernières minutes
- Filtre — durée >= 5 ET durée <= 8

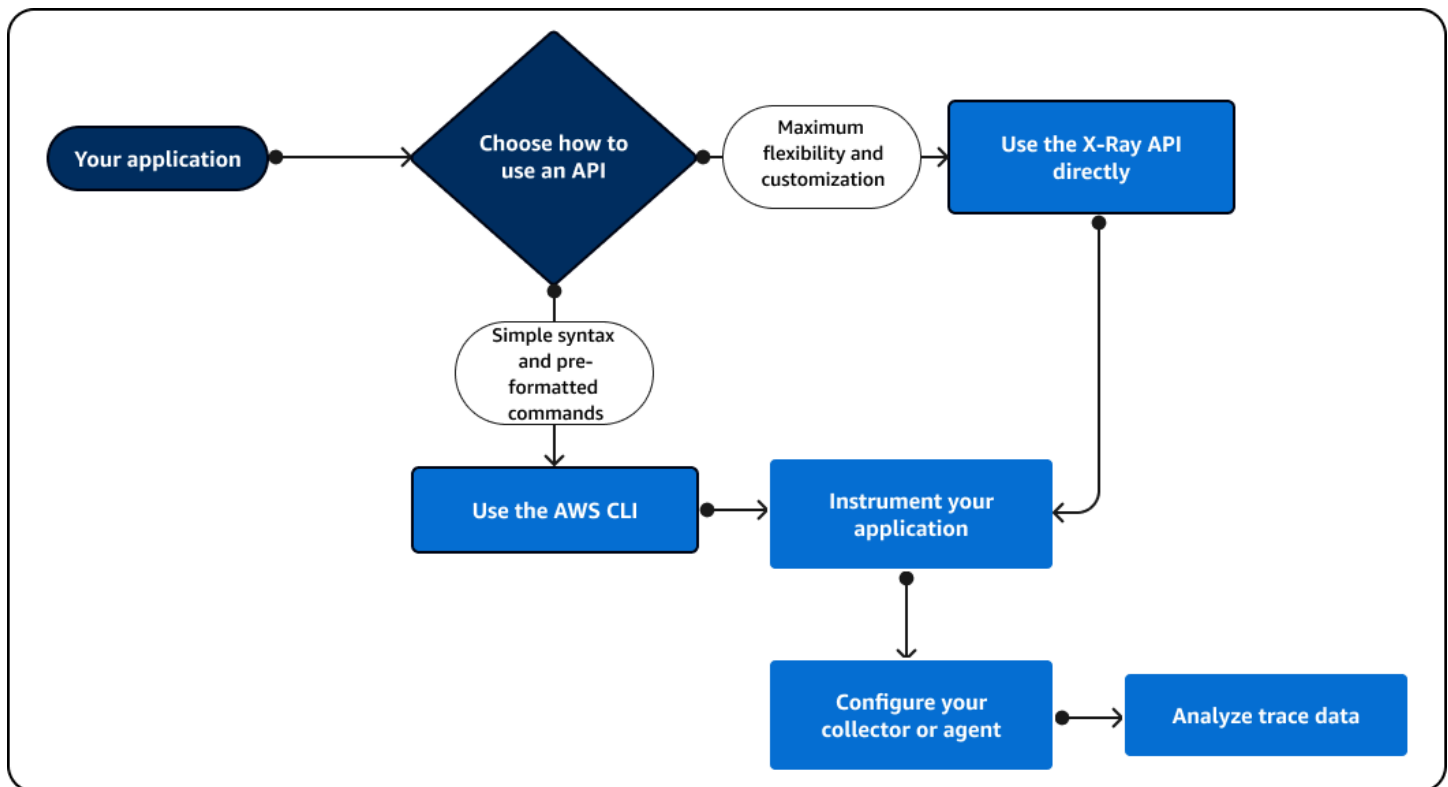
Utiliser l'API X-Ray

Si le SDK X-Ray ne prend pas en charge votre langage de programmation, vous pouvez utiliser le X-Ray APIs directement ou le AWS Command Line Interface (AWS CLI) pour appeler les commandes de l'API X-Ray. Suivez les instructions suivantes pour choisir la manière dont vous interagissez avec l'API :

- Utilisez le AWS CLI pour une syntaxe plus simple à l'aide de commandes préformatées ou d'options dans votre requête.
- Utilisez directement l'API X-Ray pour une flexibilité et une personnalisation maximales des demandes que vous envoyez à X-Ray.

Si vous utilisez directement l'[API X-Ray](#) au lieu du AWS CLI, vous devez paramétrer votre demande dans le format de données approprié et vous devrez peut-être également configurer l'authentification et la gestion des erreurs.

Le schéma suivant indique comment choisir le mode d'interaction avec l'API X-Ray :



Utilisez l'API X-Ray pour envoyer les données de suivi directement à X-Ray. L'API X-Ray prend en charge toutes les fonctions disponibles dans le SDK X-Ray, y compris les actions courantes suivantes :

- [PutTraceSegments](#)— Télécharge les documents segmentés sur X-Ray.
- [BatchGetTraces](#)— Récupère une liste de traces dans une liste de traces IDs. Chaque trace récupérée est une collection de documents segmentés provenant d'une seule demande.
- [GetTraceSummaries](#)— Récupère IDs et annote les traces. Vous pouvez spécifier a `FilterExpression` pour récupérer un sous-ensemble de résumés de traces.
- [GetTraceGraph](#)— Récupère un graphique de service pour un ID de trace spécifique.
- [GetServiceGraph](#)— Récupère un JSON document formaté qui décrit les services qui traitent les demandes entrantes et appellent les demandes en aval.

Vous pouvez également utiliser le AWS Command Line Interface (AWS CLI) dans le code de votre application pour interagir par programmation avec X-Ray. Il AWS CLI prend en charge toutes les fonctions disponibles dans le SDK X-Ray, y compris celles pour les autres Services AWS. Les fonctions suivantes sont des versions des opérations d'API répertoriées précédemment dans un format plus simple :

- [put-trace-segments](#)— Télécharge les documents segmentés sur X-Ray.
- [batch-get-traces](#)— Récupère une liste de traces dans une liste de traces IDs. Chaque trace récupérée est une collection de documents segmentés provenant d'une seule demande.
- [get-trace-summaries](#)— Récupère IDs et annote les traces. Vous pouvez spécifier a `FilterExpression` pour récupérer un sous-ensemble de résumés de traces.
- [get-trace-graph](#)— Récupère un graphique de service pour un ID de trace spécifique.
- [get-service-graph](#)— Récupère un document JSON formaté qui décrit les services qui traitent les demandes entrantes et appellent les demandes en aval.

Pour commencer, vous devez installer le [AWS CLI](#) pour votre système d'exploitation. AWS soutient Linux, macOS and Windows systèmes d'exploitation. Pour plus d'informations sur la liste des commandes X-Ray, consultez le [guide AWS CLI Command Reference pour X-Ray](#).

Rubriques

- [Utilisation de l' AWS X-Ray API avec la AWS CLI](#)
- [Envoi de données de suivi à AWS X-Ray](#)
- [Obtenir des données à partir de AWS X-Ray](#)
- [Configuration de l'échantillonnage, des groupes et des paramètres de chiffrement avec l'API AWS X-Ray](#)
- [Utilisation de règles d'échantillonnage avec l'API X-Ray](#)
- [AWS X-Ray documents segmentés](#)

Utilisation de l' AWS X-Ray API avec la AWS CLI

La AWS CLI vous permet d'accéder directement au service X-Ray et d'utiliser le même APIs que celui utilisé par la console X-Ray pour récupérer le graphe du service et les données de trace brutes. L'exemple d'application inclut des scripts qui montrent comment les utiliser APIs avec la AWS CLI.

Prérequis

Ce didacticiel utilise l'exemple d'application Scorekeep et inclut des scripts pour générer des données de suivi et une cartographie des services. Suivez les instructions du [didacticiel de mise en route](#) pour lancer l'application.

Ce didacticiel utilise le AWS CLI pour montrer l'utilisation de base de l'API X-Ray. La AWS CLI, [disponible pour Windows, Linux et OS-X](#), fournit un accès en ligne de commande au public APIs pour tous. Services AWS

Note

Vous devez vérifier que vous êtes AWS CLI configuré dans la même région que celle dans laquelle votre exemple d'application Scorekeep a été créé.

Les scripts inclus pour tester l'exemple d'application utilisent cURL pour envoyer le trafic vers l'API et jq pour analyser la sortie. Vous pouvez télécharger l'exécutable à jq partir de stedolan.github.io et l'exécutable curl à partir de <https://curl.haxx.se/download.html>. La plupart des installations Linux et OS X incluent cURL.

Génération de données de suivi

L'application web continue de générer du trafic vers l'API à intervalles réguliers de quelques secondes pendant que le jeu est en cours, mais elle génère uniquement un type de demande. Utilisez le script `test-api.sh` pour exécuter des scénarios de bout en bout et générer plusieurs autres données de suivi pendant que vous testez l'API.

Pour utiliser le script **test-api.sh**

1. Ouvrez la [console Elastic Beanstalk](#).
2. Accédez à la [console de gestion](#) de votre environnement.
3. Copiez l'URL de l'environnement à partir de l'en-tête de page.
4. Ouvrez `bin/test-api.sh` et remplacez la valeur pour l'API par l'URL de votre environnement.

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

5. Exécutez le script pour générer du trafic vers l'API.

```
~/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","EG4BPR0Q","V4BLIDJ3","9R

```

Utiliser l'API X-Ray

La AWS CLI fournit des commandes pour toutes les actions d'API proposées par X-Ray, y compris [GetServiceGraph](#) et [GetTraceSummaries](#). Consultez le document [Référence d'API AWS X-Ray](#) pour plus d'informations sur toutes les actions prises en charge et les types de données utilisés.

Exemple bin/service-graph.sh

```

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

```

Le script récupère un graphique de services pour les 10 dernières minutes.

```

~/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 '

```

Le script récupère les URLs traces générées il y a une à deux minutes.

```

~/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"
]

```

Exemple 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[*]'
```

Le script récupère les suivis entiers générés il y a une et deux minutes.

```
~/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
        \":\"388GEROC4PCA6D59ED3CTI5EEJV4KQNS05AEMVJF66Q9ASUAAJG\", \"resource_names\":
        [\"scorekeep-game-xray\"]},\"origin\":\"AWS::DynamoDB::Table\"}"
      }
    ],
    "Id": "1-5828d9f2-a90669393f4343211bc1cf75",
    "Duration": 0.05099987983703613
  }
  ...
```

Nettoyage

Mettez fin à votre environnement Elastic Beanstalk pour arrêter les instances EC2 Amazon, les tables DynamoDB et les autres ressources.

Pour arrêter votre environnement Elastic Beanstalk

1. Ouvrez la [console Elastic Beanstalk](#).
2. Accédez à la [console de gestion](#) de votre environnement.
3. Choisissez Actions.
4. Choisissez Terminate Environment (Terminer l'environnement).
5. Sélectionnez Résilier.

Les données de suivi sont automatiquement supprimées de X-Ray au bout de 30 jours.

Envoi de données de suivi à AWS X-Ray

Vous pouvez envoyer des données de suivi à X-Ray sous forme de documents segmentés. Un document de segment est une chaîne au format JSON contenant des informations sur les tâches effectuées par votre application pour répondre à une demande. Votre application peut enregistrer des données sur les tâches qu'elle effectue elle-même dans les segments, ou sur celles qui font appel à des ressources et services en aval dans les sous-segments.

Les segments enregistrent les informations sur les tâches effectuées par votre application. Au minimum, un segment enregistre le temps passé sur une tâche, un nom, IDs etc. L'ID de suivi permet de suivre la demande lors de son cheminement entre les différents services. L'ID de segment permet de suivre les tâches effectuées par un service en particulier pour répondre à la demande.

Exemple Segment terminé minimal

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


Lors de la réception d'une demande, vous pouvez envoyer un segment en cours comme espace réservé en attendant la réponse.

Exemple Segment en cours

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

Vous pouvez envoyer des segments à X-Ray directement [PutTraceSegments](#), avec ou [via le daemon X-Ray](#).

La plupart des applications appellent d'autres services ou accèdent à des ressources à l'aide du AWS SDK. Enregistrez les informations relatives aux appels en aval dans des sous-segments. X-Ray utilise des sous-segments pour identifier les services en aval qui n'envoient pas de segments et créer des entrées pour eux sur le graphique des services.

Un sous-segment peut être intégré dans un document de segment entier, ou envoyé séparément. Envoyez des sous-segments séparément pour suivre de manière asynchrone les appels en aval pour les demandes de longue durée ou pour éviter de dépasser la taille maximale du document de segment (64 kB).

Exemple Sous-segment

Un sous-segment possède un type de subsegment et un `parent_id` qui identifie le segment parent.

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

Pour plus d'informations sur les champs et les valeurs que vous pouvez inclure dans les segments et les sous-segments, consultez [AWS X-Ray documents segmentés](#).

Sections

- [Génération d'une trace IDs](#)
- [En utilisant PutTraceSegments](#)
- [Envoi de documents segmentés au daemon X-Ray](#)

Génération d'une trace IDs

Pour envoyer des données à X-Ray, vous devez générer un identifiant de trace unique pour chaque demande.

Format d'identification X-Ray Trace

Un X-Ray `trace_id` est composé de trois chiffres séparés par des tirets. Par exemple, `1-58406520-a006649127e371903a2de979`. Cela consiste notamment à :

- Le numéro de version, qui est 1.
- L'heure de la demande d'origine sous Unix Epoch Time en utilisant 8 chiffres hexadécimaux.

Par exemple, le 1er décembre 2016 à 10 h 00 PST est exprimé en 1480615200 secondes ou 58406520 en chiffres hexadécimaux.

- Identifiant 96 bits unique au monde pour la trace en 24 chiffres hexadécimaux.

Note

X-Ray prend désormais en charge IDs les traces créées à OpenTelemetry l'aide de tout autre framework conforme à la spécification [W3C Trace Context](#). Un ID de trace W3C doit être formaté au format X-Ray Trace ID lors de son envoi à X-Ray. Par exemple, l'ID de trace du W3C `4efaaf4d1e8720b39541901950019ee5` doit être formaté comme `1-4efaaf4d-1e8720b39541901950019ee5` lorsqu'il est envoyé à X-Ray. X-Ray Trace IDs inclut l'horodatage de la demande d'origine dans Unix Epoch Time, mais cela n'est pas obligatoire lors de l'envoi de la trace W3C au format IDs X-Ray.

Vous pouvez écrire un script pour générer un tracé X-Ray à des IDs fins de test. Voici deux exemples :

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"
```

Consultez l'exemple d'application Scorekeep pour découvrir les scripts qui créent des traces IDs et envoient des segments au daemon X-Ray.

- Python – [xray_start.py](#)
- Bash — [xray_start.sh](#)

En utilisant PutTraceSegments

Vous pouvez charger des documents relatifs aux segments avec l'API [PutTraceSegments](#). L'API possède un seul paramètre, `TraceSegmentDocuments`, qui accepte une liste de documents de segment au format JSON.

Avec l'AWS CLI, utilisez la `aws xray put-trace-segments` commande pour envoyer les documents de segment directement à X-Ray.

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

```
"UnprocessedTraceSegments": []  
}
```

Note

Le processeur de commandes Windows et Windows PowerShell ont des exigences différentes en matière de guillemets et d'échappements aux guillemets dans les chaînes JSON. Pour plus d'informations, consultez [Indication des chaînes entre guillemets](#) dans le Guide de l'utilisateur de l' AWS CLI .

La sortie répertorie tous les segments dont le traitement a échoué. Par exemple, si la date qui figure dans l'ID de suivi remonte à trop loin, une erreur similaire à la suivante s'affiche :

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

Vous pouvez indiquer plusieurs documents de segment à la suite, en les séparant par des espaces.

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

Envoi de documents segmentés au daemon X-Ray

Au lieu d'envoyer des documents de segment à l'API X-Ray, vous pouvez envoyer des segments et des sous-segments au daemon X-Ray, qui les mettra en mémoire tampon et les téléchargera par lots vers l'API X-Ray. Le SDK X-Ray envoie des documents segmentés au daemon afin d'éviter de passer des appels directs à AWS celui-ci.

Note

Consultez [Exécution du daemon X-Ray en local](#) pour obtenir des instructions sur l'exécution du démon.

Envoyez le segment au format JSON sur le port UDP 2000, en ajoutant en préfixe l'en-tête du démon, `{"format": "json", "version": 1}\n`

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

Sous Linux, vous pouvez envoyer des documents de segment au démon depuis un terminal Bash. Enregistrez l'en-tête et le document de segment comme fichier texte et placez-les dans `/dev/udp` avec `cat`.

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

Exemple `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"}
```

Consultez le [journal du démon](#) pour vérifier qu'il a envoyé le segment à X-Ray.

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

Obtenir des données à partir de AWS X-Ray

AWS X-Ray traite les données de suivi que vous lui envoyez pour générer des traces complètes, des résumés de traces et des graphiques de service au format JSON. Vous pouvez récupérer les données générées directement depuis l'API à l'aide de la AWS CLI.

Sections

- [Récupération du graphique de services](#)
- [Récupération du graphique des services par groupe](#)
- [Récupération des suivis](#)
- [Extraction et affinage de l'analyse de la cause racine](#)

Récupération du graphique de services

Vous pouvez utiliser l'API [GetServiceGraph](#) pour récupérer le graphique de services JSON. L'API nécessite une heure de début et de fin, que vous pouvez calculer à partir d'un terminal Linux avec la commande `date`.

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

`date +%s` détermine une date en quelques secondes. Utilisez cette valeur comme une heure de fin et déduisez le temps de traitement pour obtenir une heure de début.

Exemple Script permettant de récupérer un graphique de services pour les 10 dernières minutes

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

L'exemple suivant montre un graphe de service avec 4 nœuds, dont un nœud client, une EC2 instance, une table DynamoDB et une rubrique Amazon SNS.

Exemple `GetServiceGraph` sortie

```
{
  "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": 0.12
    },
    "ResponseTimeHistogram": [
      {
        "Value": 0.076,
        "Count": 1
      },
      {
        "Value": 0.044,
        "Count": 1
      }
    ],
    "Aliases": []
  }
]
```

```

        "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
    }
  ]
}

```

Récupération du graphique des services par groupe

Pour appeler un graphique des services en fonction du contenu d'un groupe, incluez un élément `groupName` ou `groupARN`. L'exemple suivant illustre un appel de graphique des services à un groupe nommé `Example1`.

Exemple Script permettant de récupérer un graphique des services par nom pour le groupe `Example1`

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

Récupération des suivis

Vous pouvez utiliser l'API [GetTraceSummaries](#) pour obtenir la liste des synthèses de suivi. Les résumés de traces incluent des informations que vous pouvez utiliser pour identifier les traces que vous souhaitez télécharger dans leur intégralité, notamment les annotations, les informations de demande et de réponse, et. IDs

Il y a deux indicateurs `TimeRangeType` disponibles lors de l'appel de `aws xray get-trace-summaries` :

- **TraceId**— La `GetTraceSummaries` recherche par défaut utilise l'heure de `TraceId` et renvoie les traces démarrées dans la plage calculée [`start_time`, `end_time`). Cette plage d'horodatages est calculée en fonction du codage de l'horodatage dans le `TraceId`, ou peut être définie manuellement.
- **Durée de l'événement** : pour rechercher les événements tels qu'ils se produisent au fil du temps, AWS X-Ray permet de rechercher des traces à l'aide des horodatages des événements. L'heure de l'événement renvoie les traces actives au cours de la plage [`start_time`, `end_time`), quel que soit le moment où la trace a commencé.

Utilisez la commande `aws xray get-trace-summaries` pour obtenir la liste des synthèses de suivi. Les commandes suivantes permettent d'obtenir une liste de résumés de traces réalisés il y a 1 à 2 minutes en utilisant l' `TraceId` heure par défaut.

Exemple Script permettant d'obtenir des synthèses de suivi

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

Exemple `GetTraceSummaries` sortie

```
{
  "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": "0la"
                }
            }
        ]
    },
    "ResponseTime": 3.232,
    "Duration": 3.232,

```

```

        "Id": "1-59602603-23fc5b688855d396af79b496",
        "HasThrottle": false
    }
],
"ApproximateTime": 1499473304.0,
"TracesProcessedCount": 2
}

```

Utilisez l'ID de suivi de la sortie pour obtenir un suivi complet avec l'API [BatchGetTraces](#).

Exemple BatchGetTraces commande

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

Exemple BatchGetTraces sortie

```

{
  "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\":\"01a\"}}, {\"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": []
}

```

Le suivi complet inclut un document pour chaque segment, compilé à partir de l'ensemble des documents de segments reçus pour le même ID de suivi. Ces documents ne représentent pas les données telles qu'elles ont été envoyées à X-Ray par votre application. Ils représentent plutôt les documents traités générés par le service X-Ray. X-Ray crée le document de trace complet en compilant les documents de segment envoyés par votre application et en supprimant les données non conformes au [schéma du document de segment](#).

X-Ray crée également des segments déduits pour les appels en aval vers des services qui n'envoient pas de segments eux-mêmes. Par exemple, lorsque vous appelez DynamoDB avec un client instrumenté, le SDK X-Ray enregistre un sous-segment contenant des détails sur l'appel du point de vue de celui-ci. Cependant, DynamoDB n'envoie pas de segment correspondant. X-Ray utilise les informations du sous-segment pour créer un segment inféré représentant la ressource DynamoDB dans la carte de trace, et l'ajoute au document de trace.

Pour obtenir plusieurs traces à partir de l'API, vous avez besoin d'une liste de traces IDs, que vous pouvez extraire du résultat à l'get-trace-summaries aide d'une [AWS CLI requête](#). Redirigez la liste vers l'entrée de batch-get-traces pour obtenir des traces complètes pour une période spécifique.

Exemple Script permettant d'obtenir les suivis complets pour une période d'une minute

```

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[*]'
```

Extraction et affinage de l'analyse de la cause racine

Lors de la génération d'un résumé de trace avec l'[GetTraceSummaries API](#), les résumés de suivi partiels peuvent être réutilisés dans leur format JSON pour créer une expression de filtre affinée basée sur les causes premières. Consultez les exemples ci-dessous pour connaître les étapes d'affinage.

Exemple Exemple GetTraceSummaries de sortie : section sur la cause première du temps de réponse

```
{
  "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
      }
    ]
  }
]
}

```

En modifiant la sortie ci-dessus ou en lui appliquant des omissions, ce fichier JSON peut devenir un filtre pour les entités de cause racine correspondantes. Pour chaque champ de cette sortie JSON, les correspondances de candidat doivent être exactes pour que le suivi soit renvoyé. Les champs supprimés deviennent des valeurs de caractères génériques, ce qui correspond à un format compatible avec la structure de requête de l'expression de filtre.

Exemple Cause racine des temps de réponse reformatée

```

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

```

Ce format JSON est ensuite utilisé en tant que partie d'une expression de filtre via un appel à `rootcause.json = #[{}]`. Consultez le chapitre sur les [expressions de filtre](#) pour en savoir plus sur l'exécution de requêtes avec des expressions de filtre.

Exemple Exemple de filtre JSON

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

Configuration de l'échantillonnage, des groupes et des paramètres de chiffrement avec l'API AWS X-Ray

AWS X-Ray permet APIs de configurer les [règles d'échantillonnage](#), les règles de groupe et les [paramètres de chiffrement](#).

Sections

- [Paramètres de chiffrement](#)
- [Règles d'échantillonnage](#)
- [Groups](#)

Paramètres de chiffrement

Permet [PutEncryptionConfig](#) de spécifier une clé AWS Key Management Service (AWS KMS) à utiliser pour le chiffrement.

Note

X-Ray ne prend pas en charge les clés KMS asymétriques.

```
$ 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"  
  }  
}
```

Pour l'ID de clé, vous pouvez utiliser un alias (comme illustré dans l'exemple), un ID de clé ou un ARN.

Utilisez [GetEncryptionConfig](#) pour obtenir la configuration actuelle. Lorsque X-Ray a fini d'appliquer vos paramètres, le statut passe de UPDATING à ACTIVE.

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

Pour arrêter d'utiliser une clé KMS et utiliser le chiffrement par défaut, définissez le type de chiffrement sur NONE.

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

Règles d'échantillonnage

Vous pouvez gérer les [règles d'échantillonnage](#) de votre compte à l'aide de l'API X-Ray. Pour plus d'informations sur l'ajout et la gestion de balises, consultez [Étiquetage des règles et des groupes d'échantillonnage aux rayons X](#).

Obtenez toutes les règles d'échantillonnage avec [GetSamplingRules](#).

```
$ 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
}
]
}

```

Par défaut, la règle s'applique à toutes les demandes qui ne correspondent à aucune autre règle. Elle est la règle avec la plus faible priorité et ne peut pas être supprimée. Toutefois, vous pouvez modifier la fréquence et la taille du réservoir avec [UpdateSamplingRule](#).

Exemple Entrée d'API pour [UpdateSamplingRule](#)— 10000-default.json

```

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

```

L'exemple suivant utilise le fichier précédente en tant qu'entrée pour modifier la règle par défaut sur un pour cent (1 %) sans réservoir. Les balises sont facultatives. Si vous choisissez d'ajouter des balises, une clé de balise est requise et les valeurs des balises sont facultatives. Pour supprimer des balises existantes d'une règle d'échantillonnage, utilisez [UntagResource](#)

```

$ aws xray update-sampling-rule --cli-input-json file://1000-default.json --tags
[{"Key": "key_name", "Value": "value"}, {"Key": "key_name", "Value": "value"}]
{
  "SamplingRuleRecords": [

```

```
{
  "SamplingRule": {
    "RuleName": "Default",
    "RuleARN": "arn:aws:xray:us-east-2:123456789012:sampling-rule/Default",
    "ResourceARN": "*",
    "Priority": 10000,
    "FixedRate": 0.01,
    "ReservoirSize": 0,
    "ServiceName": "*",
    "ServiceType": "*",
    "Host": "*",
    "HTTPMethod": "*",
    "URLPath": "*",
    "Version": 1,
    "Attributes": {}
  },
  "CreatedAt": 0.0,
  "ModifiedAt": 1529959993.0
},
```

Créez les règles d'échantillonnage supplémentaires avec [CreateSamplingRule](#). Lorsque vous créez une règle, la plupart des champs de la règle sont obligatoires. L'exemple suivant crée deux règles. Cette première règle définit un taux de base pour l'exemple d'application Scorekeep. Elle correspond à toutes les demandes servies par l'API qui ne correspond pas à la priorité de la règle.

Exemple Entrée d'API pour [UpdateSamplingRule](#)— 9000-base-scorekeep.json

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

La deuxième règle s'applique également à Scorekeep, mais elle a une priorité plus élevée et est plus spécifique. Cette règle définit un très faible taux d'échantillonnage pour les demandes d'interrogation. Il s'agit de demandes GET effectuées par le client toutes les deux ou trois secondes pour rechercher les modifications apportées à l'état du jeu.

Exemple Entrée d'API pour [UpdateSamplingRule](#)— 5000-polling-scorekeep.json

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

Les balises sont facultatives. Si vous choisissez d'ajouter des balises, une clé de balise est requise et les valeurs des balises sont facultatives.

```
$ 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
    }
}

```

Pour supprimer une règle d'échantillonnage, utilisez [DeleteSamplingRule](#).

```

$ aws xray delete-sampling-rule --rule-name polling-scorekeep
{
    "SamplingRuleRecord": {
        "SamplingRule": {
            "RuleName": "polling-scorekeep",
            "RuleARN": "arn:aws:xray:us-east-1:123456789012:sampling-rule/polling-
scorekeep",
            "ResourceARN": "*",
            "Priority": 5000,

```

```

        "FixedRate": 0.003,
        "ReservoirSize": 0,
        "ServiceName": "Scorekeep",
        "ServiceType": "*",
        "Host": "*",
        "HTTPMethod": "GET",
        "URLPath": "/api/state/*",
        "Version": 1,
        "Attributes": {}
    },
    "CreatedAt": 1530574399.0,
    "ModifiedAt": 1530574399.0
}

```

Groups

Vous pouvez utiliser l'API X-Ray pour gérer les groupes de votre compte. Les groupes constituent un ensemble de suivis qui sont définis par une expression de filtre. Vous pouvez utiliser des groupes pour générer des graphiques de service supplémentaires et fournir CloudWatch des statistiques Amazon. Consultez [Obtenir des données à partir de AWS X-Ray](#) pour plus de détails sur l'utilisation des graphiques et des métriques de service via l'API X-Ray. Pour plus d'informations sur les groupes, consultez [Configuration des groupes](#). Pour plus d'informations sur l'ajout et la gestion de balises, consultez [Étiquetage des règles et des groupes d'échantillonnage aux rayons X](#).

Création d'un groupe avec CreateGroup Les balises sont facultatives. Si vous choisissez d'ajouter des balises, une clé de balise est requise et les valeurs des balises sont facultatives.

```

$ 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}"
}

```

Obtenez tous les groupes existants à l'aide de GetGroups.

```

$ 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"
  }

```

Mettez à jour un groupe à l'aide de `UpdateGroup`. Les balises sont facultatives. Si vous choisissez d'ajouter des balises, une clé de balise est requise et les valeurs des balises sont facultatives. Pour supprimer des balises existantes d'un groupe, utilisez [UntagResource](#).

```

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

```

Supprimez un groupe à l'aide de `DeleteGroup`.

```

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

```

Utilisation de règles d'échantillonnage avec l'API X-Ray

Le AWS X-Ray SDK utilise l'API X-Ray pour obtenir des règles d'échantillonnage, communiquer les résultats d'échantillonnage et obtenir des quotas. Vous pouvez les utiliser APIs pour mieux

comprendre le fonctionnement des règles d'échantillonnage ou pour implémenter l'échantillonnage dans un langage non pris en charge par le kit de développement X-Ray.

Commencez par obtenir toutes les règles d'échantillonnage avec [GetSamplingRules](#).

```
$ 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
    }
]
}

```

La sortie inclut la règle par défaut et les règles personnalisées. Consultez [Règles d'échantillonnage](#) si vous n'avez pas encore créé de règles d'échantillonnage.

Évaluez les règles par rapport aux demandes entrantes par ordre croissant de priorité. Lorsqu'une règle correspond, utilisez la fréquence et la taille de réservoir fixées pour prendre une décision d'échantillonnage. Enregistrez les demandes échantillonnées et ignorez (à des fins de suivi) les demandes non échantillonnées. Arrêtez l'évaluation des règles lorsqu'une décision d'échantillonnage est prise.

Une taille de réservoir de règles est le nombre cible de suivis à enregistrer par seconde avant d'appliquer la fréquence fixe. Le réservoir s'applique à tous les services cumulativement. Vous ne pouvez donc pas l'utiliser directement. Toutefois, s'il est différent de zéro, vous pouvez emprunter une trace par seconde dans le réservoir jusqu'à ce que X-Ray vous assigne un quota. Avant de recevoir un quota, enregistrez la première demande chaque seconde et appliquez le taux fixe aux demandes additionnelles. La fréquence fixe est un nombre décimal compris entre 0 et 1,00 (100 %).

L'exemple suivant illustre un appel à [GetSamplingTargets](#) avec les détails sur les décisions d'échantillonnage effectuées au cours des 10 dernières secondes.

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

La réponse de X-Ray inclut un quota à utiliser au lieu d'emprunter au réservoir. Dans cet exemple, le service a emprunté 10 suivis au réservoir sur 10 secondes et appliqué la fréquence fixe de 10 % aux 100 autres demandes, ce qui se traduit par un total de 20 demandes échantillonnées. Le quota est valable pendant cinq minutes (indiqué par l'heure de vie) ou jusqu'à ce qu'un nouveau quota soit attribué. X-Ray peut également attribuer un intervalle de rapport plus long que celui par défaut, bien que ce ne soit pas le cas ici.

Note

La réponse de X-Ray n'inclut peut-être pas de quota la première fois que vous l'appellez. Continuez à emprunter au réservoir jusqu'à ce qu'un quota vous soit attribué.

Les deux autres champs de la réponse peuvent indiquer des problèmes avec l'entrée. Vérifiez `LastRuleModification` par rapport à votre dernier appel de [GetSamplingRules](#). S'il est plus récent, obtenez une nouvelle copie des règles. `UnprocessedStatistics` peut inclure des erreurs qui indiquent qu'une règle a été supprimée, que les statistiques de document dans l'entrée étaient trop anciennes ou qu'elle comportait des erreurs d'autorisation.

AWS X-Ray documents segmentés

Un segment de suivi est une représentation JSON d'une demande que sert votre application. Un segment de trace enregistre des informations sur la demande d'origine, des informations sur le travail effectué localement par votre application et des sous-segments contenant des informations sur les appels en aval que votre application effectue vers les AWS ressources, les bases de données HTTP APIs et SQL.

Un document de segment transmet des informations sur un segment à X-Ray. Un document segmenté peut peser jusqu'à 64 kB et contenir un segment entier avec des sous-segments, un fragment de segment indiquant qu'une demande est en cours ou un sous-segment unique envoyé séparément. Vous pouvez envoyer des documents de segment directement à X-Ray à l'aide de l'[PutTraceSegments](#) API.

X-Ray compile et traite des documents segmentés pour générer des résumés de traces interrogeables et des traces complètes auxquels vous pouvez accéder en utilisant respectivement le [GetTraceSummaries](#) et [BatchGetTraces](#) APIs. Outre les segments et sous-segments que vous envoyez à X-Ray, le service utilise les informations contenues dans les sous-segments pour générer des segments déduits et les ajouter à la trace complète. Les segments déduits représentent les services et les ressources en aval sur la carte de suivi.

X-Ray fournit un schéma JSON pour les documents segmentés. Vous pouvez télécharger le schéma ici : [xray-segmentdocument-schema-v1.0.0](#). Les champs et les objets figurant dans le schéma sont décrits plus en détail dans les sections suivantes.

Un sous-ensemble de champs de segment est indexé par X-Ray pour être utilisé avec des expressions de filtre. Par exemple, si vous attribuez un identifiant unique au `user` champ d'un segment, vous pouvez rechercher des segments associés à des utilisateurs spécifiques dans la console X-Ray ou à l'aide de l'`GetTraceSummariesAPI`. Pour de plus amples informations, veuillez consulter [Utilisation d'expressions de filtre](#).

Lorsque vous instrumentez votre application avec le SDK X-Ray, celui-ci génère des documents segmentés pour vous. Au lieu d'envoyer les documents segmentés directement à X-Ray, le SDK les transmet via un port UDP local au daemon [X-Ray](#). Pour de plus amples informations, veuillez consulter [Envoi de documents segmentés au daemon X-Ray](#).

Sections

- [Champs de segment](#)
- [Sous-segments](#)
- [Données de demande HTTP](#)
- [Annotations](#)
- [Métadonnées](#)
- [AWS données sur les ressources](#)
- [Erreurs et exceptions](#)
- [Requêtes SQL](#)

Champs de segment

Un segment enregistre les informations de suivi d'une demande que votre application sert. Au minimum, un segment enregistre le nom, l'ID, l'heure de début, l'ID de suivi et l'heure de fin de la demande.

Exemple Segment terminé minimal

```
{
  "name" : "example.com",
  "id" : "70de5b6f19ff9a0a",
```



```
"start_time" : 1.478293361271E9,  
"trace_id" : "1-581cf771-a006649127e371903a2de979",  
"end_time" : 1.478293361449E9  
}
```

Les champs suivants sont obligatoires, ou soumis à condition, pour les segments.

 Note

Les valeurs doivent être des chaînes (jusqu'à 250 caractères), sauf mention contraire.

Champs de segment obligatoires

- **name**— Nom logique du service qui a traité la demande, 200 caractères maximum. Par exemple, le nom de votre application ou le nom de domaine. Les noms peuvent contenir des lettres en Unicode, des nombres et des espaces, ainsi que les symboles suivants : `_`, `.`, `:`, `/`, `%`, `&`, `#`, `=`, `+`, `\`, `-`, `@`
- **id**— Un identifiant 64 bits pour le segment, unique parmi les segments d'une même trace, en 16 chiffres hexadécimaux.
- **trace_id**— Un identifiant unique qui connecte tous les segments et sous-segments issus d'une seule demande client.

Format d'identification X-Ray Trace

Un X-Ray `trace_id` est composé de trois chiffres séparés par des tirets. Par exemple, 1-58406520-a006649127e371903a2de979. Cela consiste notamment à :

- Le numéro de version, qui est 1.
- L'heure de la demande initiale sous Unix Epoch Time en utilisant 8 chiffres hexadécimaux.

Par exemple, le 1er décembre 2016 à 10 h 00 PST est exprimé en 1480615200 secondes ou 58406520 en chiffres hexadécimaux.

- Identifiant 96 bits unique au monde pour la trace en 24 chiffres hexadécimaux.

 Note

X-Ray prend désormais en charge IDs les traces créées à OpenTelemetry l'aide de tout autre framework conforme à la spécification [W3C Trace Context](#). Un identifiant de trace

W3C doit être formaté au format X-Ray trace ID lors de son envoi à X-Ray. Par exemple, l'ID de trace du W3C 4efaaf4d1e8720b39541901950019ee5 doit être formaté comme 1-4efaaf4d-1e8720b39541901950019ee5 lorsqu'il est envoyé à X-Ray. X-Ray Trace IDs inclut l'horodatage de la demande d'origine dans Unix Epoch Time, mais cela n'est pas obligatoire lors de l'envoi de la trace W3C au format IDs X-Ray.

Sécurité de l'ID de suivi

IDs Les traces sont visibles dans les [en-têtes de réponse](#). Générez des traces IDs à l'aide d'un algorithme aléatoire sécurisé pour empêcher les attaquants de calculer les traces futures IDs et d'envoyer des demandes avec celles-ci IDs à votre application.

- `start_time`— nombre correspondant à l'heure à laquelle le segment a été créé, en secondes à virgule flottante par rapport à l'époque. Par exemple, 1480615200.010 ou 1.480615200010E9. Utilisez autant de décimales que nécessaire. La résolution en microsecondes est recommandée si elle est disponible.
- `end_time`— numéro correspondant à l'heure à laquelle le segment a été fermé. Par exemple, 1480615200.090 ou 1.480615200090E9. Spécifiez une `end_time` ou `in_progress`.
- `in_progress`— booléen, défini sur `true` lieu de spécifier un `end_time` pour enregistrer qu'un segment est démarré mais qu'il n'est pas terminé. Envoyez un segment en cours lorsque votre application reçoit une demande longue à servir, pour suivre la réception de la demande. Lorsque la réponse est envoyée, envoyez le segment terminé pour remplacer le segment en cours. N'envoyez qu'un segment complet, et un ou zéro segment en cours, par demande.

Noms des services

Un segment name doit correspondre au nom de domaine ou au nom logique du service qui génère le segment. Toutefois, cela n'est pas appliqué. Toute application autorisée [PutTraceSegments](#) peut envoyer des segments sous n'importe quel nom.

Les champs suivants sont facultatifs pour les segments.

Champs de segment facultatifs

- `service`— Un objet contenant des informations sur votre application.

- `version`— Chaîne identifiant la version de votre application qui a répondu à la demande.
- `user`— Chaîne identifiant l'utilisateur qui a envoyé la demande.
- `origin`— Type de AWS ressource exécutant votre application.

Valeurs prises en charge

- `AWS::EC2::Instance`— Une EC2 instance Amazon.
- `AWS::ECS::Container`— Un conteneur Amazon ECS.
- `AWS::ElasticBeanstalk::Environment`— Un environnement Elastic Beanstalk.

Lorsque plusieurs valeurs sont applicables à votre application, utilisez celle qui est la plus spécifique. Par exemple, un environnement Docker Elastic Beanstalk multicontainer exécute votre application sur un conteneur Amazon ECS, qui s'exécute à son tour sur une instance Amazon. EC2 Dans ce cas, vous devez définir l'origine sur `AWS::ElasticBeanstalk::Environment` car l'environnement est le parent des deux autres ressources.

- `parent_id`— Un ID de sous-segment que vous spécifiez si la demande provient d'une application instrumentée. Le SDK X-Ray ajoute l'ID du sous-segment parent à l'[en-tête de suivi](#) pour les appels HTTP en aval. Dans le cas de sous-segments imbriqués, un sous-segment peut avoir un segment ou un sous-segment comme parent.
- `http`— [http](#)objets contenant des informations sur la requête HTTP d'origine.
- `aws`— [aws](#)objet contenant des informations sur la AWS ressource sur laquelle votre application a envoyé la demande.
- `error`, `throttle`, `fault`, et `cause` — champs [d'erreur](#) qui indiquent qu'une erreur s'est produite et qui incluent des informations sur l'exception à l'origine de l'erreur.
- `annotations`— [annotations](#)objet avec des paires clé-valeur que vous souhaitez que X-Ray indexe pour la recherche.
- `metadata`: [metadata](#)objet contenant toutes les données supplémentaires que vous souhaitez stocker dans le segment.
- `subsegments`— ensemble d'[subsegment](#)objets.

Sous-segments

Vous pouvez créer des sous-segments pour enregistrer les appels Services AWS et les ressources que vous effectuez avec le AWS SDK, les appels vers le Web APIs HTTP interne ou externe ou les requêtes de base de données SQL. Vous pouvez également créer des sous-segments pour

déboguer ou annoter les blocs de code de votre application. Comme les sous-segments peuvent contenir d'autres sous-segments, un sous-segment personnalisé qui enregistre les métadonnées relatives à un appel de fonction interne peut contenir d'autres sous-segments personnalisés et les sous-segments d'appels en aval.

Un sous-segment enregistre un appel en aval du point de vue du service qui l'appelle. X-Ray utilise des sous-segments pour identifier les services en aval qui n'envoient pas de segments et créer des entrées pour eux sur le graphique des services.

Un sous-segment peut être intégré dans un document de segment entier, ou envoyé séparément. Envoyez des sous-segments séparément aux appels en aval de suivi de manière asynchrone pour les demandes de longue durée, ou pour éviter de dépasser la taille du document de segment maximum.

Exemple Segment avec sous-segment intégré

Un sous-segment indépendant possède `type` comme `subsegment` et un `parent_id` qui identifie le segment parent.

```
{
  "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
      }
    }
  }
]
}

```

Pour les demandes de longue durée, vous pouvez envoyer un segment en cours pour informer X-Ray de la réception de la demande, puis envoyer des sous-segments séparément pour les suivre avant de terminer la demande initiale.

Exemple Segment en cours

```

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

```

Exemple Sous-segment indépendant

Un sous-segment indépendant possède type comme subsegment, un trace_id et un parent_id qui identifie le segment parent.

```

{
  "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
  }
}
```

Lorsque la demande est terminée, fermez le segment en le renvoyant avec un `end_time`. Le segment complet remplace le segment en cours.

Vous pouvez également envoyer des sous-segments séparément pour les demandes complètes qui ont déclenché des workflows asynchrones. Par exemple, une API Web peut renvoyer une réponse OK 200 juste avant de démarrer le travail que l'utilisateur demandé. Vous pouvez envoyer un segment complet à X-Ray dès que la réponse est envoyée, suivi de sous-segments pour le travail terminé ultérieurement. Comme pour les segments, vous pouvez aussi envoyer un fragment de sous-segment pour enregistrer le démarrage du sous-segment, puis le remplacer par un sous-segment complet une fois que l'appel en aval est terminé.

Les champs suivants sont obligatoires, ou soumis à condition, pour les sous-segments.

Note

Les valeurs doivent être des chaînes (jusqu'à 250 caractères), sauf mention contraire.

Champs de sous-segment obligatoires

- **id**— Un identifiant de 64 bits pour le sous-segment, unique parmi les segments d'une même trace, en 16 chiffres hexadécimaux.
- **name**— Nom logique du sous-segment. Pour les appels en aval, nommez le sous-segment après l'appel de la ressource ou du service. Pour les sous-segments personnalisés, nommez le sous-segment après le code qu'il instrumente (par exemple, un nom de fonction).

- `start_time`— nombre correspondant à l'heure à laquelle le sous-segment a été créé, en secondes à virgule flottante par rapport à l'époque, avec une précision de quelques millisecondes. Par exemple, 1480615200.010 ou 1.480615200010E9.
- `end_time`— numéro correspondant à l'heure à laquelle le sous-segment a été fermé. Par exemple, 1480615200.090 ou 1.480615200090E9. Spécifiez une `end_time` ou `in_progress`.
- `in_progress`— booléen défini sur `true` au lieu de spécifier un `end_time` pour enregistrer qu'un sous-segment est démarré, mais qu'il n'est pas terminé. Envoyez uniquement un sous-segment complet, et un ou zéro sous-segment en cours, par demande en aval.
- `trace_id`— ID de trace du segment parent du sous-segment. Obligatoire uniquement en cas d'envoi séparé d'un sous-segment.

Format d'identification X-Ray Trace

Un X-Ray `trace_id` est composé de trois chiffres séparés par des tirets. Par exemple, 1-58406520-a006649127e371903a2de979. Cela consiste notamment à :

- Le numéro de version, qui est 1.
- L'heure de la demande initiale sous Unix Epoch Time en utilisant 8 chiffres hexadécimaux.

Par exemple, le 1er décembre 2016 à 10 h 00 PST est exprimé en 1480615200 secondes ou 58406520 en chiffres hexadécimaux.

- Identifiant 96 bits unique au monde pour la trace en 24 chiffres hexadécimaux.

Note

X-Ray prend désormais en charge IDs les traces créées à OpenTelemetry l'aide de tout autre framework conforme à la spécification [W3C Trace Context](#). Un identifiant de trace W3C doit être formaté au format X-Ray trace ID lors de son envoi à X-Ray. Par exemple, l'ID de trace du W3C 4efaaf4d1e8720b39541901950019ee5 doit être formaté comme 1-4efaaf4d-1e8720b39541901950019ee5 lorsqu'il est envoyé à X-Ray. X-Ray Trace IDs inclut l'horodatage de la demande d'origine dans Unix Epoch Time, mais cela n'est pas obligatoire lors de l'envoi de la trace W3C au format IDs X-Ray.

- `parent_id`— ID de segment du segment parent du sous-segment. Obligatoire uniquement en cas d'envoi séparé d'un sous-segment. Dans le cas de sous-segments imbriqués, un sous-segment peut avoir un segment ou un sous-segment comme parent.

- `type`—`subsegment`. Obligatoire uniquement si vous envoyez un sous-segment séparément.

Les champs suivants sont facultatifs pour les sous-segments.

Champs de sous-segment facultatifs

- `namespace`— `aws` pour les appels du SDK AWS ; `remote` pour les autres appels en aval.
- `http`— [http](#) objet contenant des informations relatives à un appel HTTP sortant.
- `aws`: [aws](#) objet contenant des informations sur la AWS ressource en aval appelée par votre application.
- `error`, `throttle`, `fault`, et `cause` — champs [d'erreur](#) qui indiquent qu'une erreur s'est produite et qui incluent des informations sur l'exception à l'origine de l'erreur.
- `annotations`— [annotations](#) objet avec des paires clé-valeur que vous souhaitez que X-Ray indexe pour la recherche.
- `metadata`: [metadata](#) objet contenant toutes les données supplémentaires que vous souhaitez stocker dans le segment.
- `subsegments`— ensemble d'[subsegment](#) objets.
- `precursor_ids`— tableau de sous-segments identifiant IDs les sous-segments ayant le même parent que ceux terminés avant ce sous-segment.

Données de demande HTTP

Utilisez un bloc HTTP pour enregistrer les détails relatifs à une demande HTTP que votre application a servie (dans un segment) ou que votre application a effectuée auprès d'une API HTTP en aval (dans un sous-segment). La plupart des champs de cet objet correspondent aux informations disponibles dans une demande et une réponse HTTP.

http

Tous les champs sont facultatifs.

- `request`— Informations relatives à une demande.
 - `method`— La méthode de demande. Par exemple, GET.
 - `url`— L'URL complète de la demande, compilée à partir du protocole, du nom d'hôte et du chemin de la demande.
 - `user_agent`— La chaîne d'agent utilisateur du client du demandeur.

- `client_ip`— L'adresse IP du demandeur. Peut être récupérée depuis la `Source Address` du paquet IP ou, pour les demandes acheminées, à partir d'un en-tête `X-Forwarded-For`.
- `x_forwarded_for`— (segments uniquement) booléen indiquant que le `client_ip` a été lu à partir d'un `X-Forwarded-For` en-tête et qu'il n'est pas fiable car il aurait pu être falsifié.
- `traced`— (sous-segments uniquement) booléen indiquant que l'appel en aval concerne un autre service suivi. Si ce champ est défini sur `true`, X-Ray considère que la trace est interrompue jusqu'à ce que le service en aval télécharge un segment dont le `parent_id` nom correspond à celui `id` du sous-segment contenant ce bloc.
- `response`— Informations relatives à une réponse.
 - `status`— entier indiquant le statut HTTP de la réponse.
 - `content_length`— entier indiquant la longueur du corps de la réponse en octets.

Lorsque vous instrumentez un appel à une API Web en aval, enregistrez un sous-segment contenant des informations sur la requête et la réponse HTTP. X-Ray utilise le sous-segment pour générer un segment inféré pour l'API distante.

Exemple Segment pour un appel HTTP envoyé par une application exécutée sur Amazon EC2

```
{
  "id": "6b55dcc497934f1a",
  "start_time": 1484789387.126,
  "end_time": 1484789387.535,
  "trace_id": "1-5880168b-fd5158284b67678a3bb5a78c",
  "name": "www.example.com",
  "origin": "AWS::EC2::Instance",
  "aws": {
    "ec2": {
      "availability_zone": "us-west-2c",
      "instance_id": "i-0b5a4678fc325bg98"
    },
    "xray": {
      "sdk_version": "2.11.0 for Java"
    },
  },
  "http": {
    "request": {
      "method": "POST",
      "client_ip": "78.255.233.48",
      "url": "http://www.example.com/api/user",
```

```
    "user_agent": "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:45.0) Gecko/20100101  
Firefox/45.0",  
    "x_forwarded_for": true  
  },  
  "response": {  
    "status": 200  
  }  
}
```

Exemple Sous-segment pour un appel HTTP en aval

```
{  
  "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  
    }  
  }  
}
```

Exemple Segment déduit pour un appel HTTP en aval

```
{  
  "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  
    }  
  }  
}
```

```
    "response": {
      "content_length": -1,
      "status": 200
    },
    "inferred": true
  }
```

Annotations

Les segments et sous-segments peuvent inclure un annotations objet contenant un ou plusieurs champs indexés par X-Ray pour une utilisation avec des expressions de filtre. Les champs peuvent avoir des valeurs de type chaîne, numérique ou booléen (ni objet ni tableau). X-Ray indexe jusqu'à 50 annotations par trace.

Exemple Segment pour appel HTTP avec annotations

```
{
  "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
  }
}
```

Les clés doivent être au format alphanumérique pour fonctionner avec les filtres. Les traits de soulignement sont autorisés. Les autres symboles et les espaces ne sont pas autorisés.

Métadonnées

Les segments et sous-segments peuvent inclure un metadata objet contenant un ou plusieurs champs avec des valeurs de n'importe quel type, y compris des objets et des tableaux. X-Ray n'indexe pas les métadonnées et les valeurs peuvent être de n'importe quelle taille, tant que le document segmenté ne dépasse pas la taille maximale (64 kB). Vous pouvez consulter les métadonnées dans le document de segment complet retourné par l'API [BatchGetTraces](#). Les clés de champ (debug dans l'exemple suivant) commençant par AWS. sont réservées à l'utilisation par AWS-provided SDKs et par les clients.

Exemple Sous-segment personnalisé avec métadonnées

```
{
  "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 données sur les ressources

Pour les segments, l'objet `aws` contient les informations relatives à la ressource sur laquelle votre application est en cours d'exécution. Plusieurs champs peuvent s'appliquer à une même ressource. Par exemple, une application exécutée dans un environnement Docker multiconteneur sur Elastic Beanstalk peut contenir des informations sur l'instance Amazon, le conteneur EC2 Amazon ECS exécuté sur l'instance et l'environnement Elastic Beanstalk lui-même.

aws (Segments)

Tous les champs sont facultatifs.

- `account_id`— Si votre application envoie des segments à un autre Compte AWS, enregistrez l'identifiant du compte qui exécute votre application.
- `cloudwatch_logs`— Tableau d'objets décrivant un seul groupe de CloudWatch logs.
 - `log_group`— Le nom du groupe de CloudWatch journaux.
 - `arn`— L'ARN du groupe de CloudWatch logs.
- `ec2`— Informations sur une EC2 instance Amazon.
 - `instance_id`— L'ID d'instance de l' EC2 instance.
 - `instance_size`— Type d' EC2 instance.
 - `ami_id`— L'identifiant Amazon Machine Image.

- `availability_zone`— La zone de disponibilité dans laquelle l'instance est exécutée.
- `ecs`— Informations sur un conteneur Amazon ECS.
 - `container`— Le nom d'hôte de votre conteneur.
 - `container_id`— L'identifiant complet de votre conteneur.
 - `container_arn`— L'ARN de votre instance de conteneur.
- `eks`— Informations sur un cluster Amazon EKS.
 - `pod`— Le nom d'hôte de votre pod EKS.
 - `cluster_name`— Le nom du cluster EKS.
 - `container_id`— L'identifiant complet de votre conteneur.
- `elastic_beanstalk`— Informations sur un environnement Elastic Beanstalk. Vous trouverez ces informations dans un fichier nommé `/var/elasticbeanstalk/xray/environment.conf` sur les dernières plateformes Elastic Beanstalk.
 - `environment_name` : Nom de l'environnement.
 - `version_label`— Nom de la version de l'application actuellement déployée sur l'instance qui a répondu à la demande.
 - `deployment_id`— numéro indiquant l'ID du dernier déploiement réussi sur l'instance qui a répondu à la demande.
- `xray`— Des métadonnées sur le type et la version de l'instrumentation utilisée.
 - `auto_instrumentation`— Booléen indiquant si l'instrumentation automatique a été utilisée (par exemple, l'agent Java).
 - `sdk_version`— Version du SDK ou de l'agent utilisé.
 - `sdk`— Le type de SDK.

Exemple AWS bloc avec plugins

```
"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"
  }
}

```

Pour les sous-segments, enregistrez les informations relatives aux ressources Services AWS et aux ressources auxquelles votre application accède. X-Ray utilise ces informations pour créer des segments déduits qui représentent les services en aval dans votre carte des services.

aws (Sous-segments)

Tous les champs sont facultatifs.

- **operation**— Le nom de l'action d'API invoquée contre une ressource Service AWS or.
- **account_id**— Si votre application accède aux ressources d'un autre compte ou envoie des segments vers un autre compte, enregistrez l'ID du compte propriétaire de la AWS ressource à laquelle votre application a accédé.
- **region**— Si la ressource se trouve dans une région différente de celle de votre application, enregistrez la région. Par exemple, `us-west-2`.
- **request_id**— Identifiant unique de la demande.
- **queue_url**— Pour les opérations sur une file d'attente Amazon SQS, l'URL de la file d'attente.
- **table_name**— Pour les opérations sur une table DynamoDB, nom de la table.

Exemple Sous-segment pour un appel à DynamoDB pour enregistrer un élément

```

{
  "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",
  }
}
```

Erreurs et exceptions

Lorsqu'une erreur se produit, vous pouvez enregistrer les détails de l'erreur et les exceptions qui ont été générées. Enregistrez les erreurs dans les segments lorsque votre application renvoie une erreur à l'utilisateur et dans les sous-segments lorsqu'un appel en aval renvoie une erreur.

types d'erreur

Définissez un ou plusieurs des champs suivants sur `true` pour indiquer qu'une erreur s'est produite. Plusieurs types peuvent s'appliquer en cas d'erreurs composites. Par exemple, une erreur `429 Too Many Requests` d'un appel en aval peut contraindre votre application à renvoyer `500 Internal Server Error`, auquel cas les trois types s'appliquent.

- `error`— booléen indiquant qu'une erreur client s'est produite (le code d'état de la réponse était `4XX Client Error`).
- `throttle`— booléen indiquant qu'une demande a été limitée (le code d'état de la réponse était `429 demandes trop nombreuses`).
- `fault`— booléen indiquant qu'une erreur de serveur s'est produite (le code d'état de réponse était `5XX Server Error`).

Indiquez la cause de l'erreur en incluant un objet `cause` dans le segment ou le sous-segment.

cause

Une cause peut être un ID d'exception sur 16 caractères ou un objet avec les champs suivants :

- `working_directory`— Le chemin complet du répertoire de travail lorsque l'exception s'est produite.
- `paths`— Le tableau des chemins d'accès aux bibliothèques ou aux modules utilisés lorsque l'exception s'est produite.
- `exceptions`— Le tableau des objets d'exception.

Incluez les informations détaillées sur l'erreur dans un ou plusieurs objets exception.

exception

Tous les champs sont facultatifs.

- `id`— Un identifiant de 64 bits pour l'exception, unique parmi les segments d'une même trace, en 16 chiffres hexadécimaux.
- `message`— Le message d'exception.
- `type`— Le type d'exception.
- `remote`— booléen indiquant que l'exception a été provoquée par une erreur renvoyée par un service en aval.
- `truncated`— entier indiquant le nombre de cadres de pile qui sont omis dans `stack`.
- `skipped`— entier indiquant le nombre d'exceptions ignorées entre cette exception et son enfant, c'est-à-dire l'exception qu'elle a provoquée.
- `cause`— ID d'exception du parent de l'exception, c'est-à-dire l'exception à l'origine de cette exception.
- `stack`— tableau d'objets `StackFrame`.

Le cas échéant, enregistrez les informations relatives à la pile des appels dans les objets `stackFrame`.

stackFrame

Tous les champs sont facultatifs.

- `path`— Le chemin relatif du fichier.
- `line`— La ligne du fichier.
- `label`— Le nom de la fonction ou de la méthode.

Requêtes SQL

Vous pouvez créer des sous-segments pour les requêtes que votre application effectue auprès d'une base de données SQL.

sql

Tous les champs sont facultatifs.

- `connection_string`— Pour les connexions à SQL Server ou à d'autres bases de données qui n'utilisent pas de chaînes de connexion URL, enregistrez la chaîne de connexion, à l'exception des mots de passe.
- `url`— Pour une connexion à une base de données utilisant une chaîne de connexion URL, enregistrez l'URL, à l'exception des mots de passe.
- `sanitized_query`— La requête de base de données, avec toutes les valeurs fournies par l'utilisateur supprimées ou remplacées par un espace réservé.
- `database_type`— Nom du moteur de base de données.
- `database_version`— Numéro de version du moteur de base de données.
- `driver_version`— Le nom et le numéro de version du pilote du moteur de base de données utilisé par votre application.
- `user`— Le nom d'utilisateur de la base de données.
- `preparation`— `call` si la requête a utilisé un `PreparedStatement` ; `statement` si la requête a utilisé `PreparedStatement` a.

Exemple Sous-segment avec une requête SQL

```
{
  "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 concepts

AWS X-Ray reçoit les données des services sous forme de segments. X-Ray regroupe ensuite les segments ayant une requête commune dans des traces. X-Ray traite les traces pour générer un graphe de service fournissant une représentation visuelle de votre application.

Concepts

- [Segments](#)
- [Sous-segments](#)
- [Graphique de services](#)
- [Suivis](#)
- [Echantillonnage](#)
- [En-tête de suivi](#)
- [Expressions de filtrage](#)
- [Groups](#)
- [Annotations et métadonnées](#)
- [Erreurs, anomalies et exceptions](#)

Segments

Les ressources de calcul qui exécutent la logique de votre application envoient des données concernant leur travail sous forme de segments. Chaque segment indique le nom de la ressource, des détails concernant la demande, ainsi que des détails sur le travail effectué. Par exemple, lorsqu'une demande HTTP parvient à votre application, elle peut enregistrer les données suivantes sur :

- L'hôte : nom d'hôte, alias ou adresse IP
- La demande : méthode, adresse du client, chemin, agent utilisateur
- La réponse : statut, contenu
- Le travail effectué : heures de début et de fin, sous-segments
- Problèmes qui se produisent : [erreurs, défauts et exceptions](#), y compris la capture automatique des piles d'exceptions.

Segment details: Scorekeep



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

Le SDK X-Ray collecte des informations à partir des en-têtes de demande et de réponse, du code de votre application et des métadonnées relatives aux AWS ressources sur lesquelles elle s'exécute. Vous choisissez les données à collecter en modifiant la configuration ou le code de votre application afin d'instrumenter les demandes entrantes, les demandes en aval et les clients du AWS SDK.

Demandes transmises

Si un équilibreur de charge ou un autre intermédiaire transmet une demande à votre application, X-Ray prend l'adresse IP du client depuis l'`X-Forwarded-For`-en-tête de la demande plutôt que depuis l'adresse IP source du paquet IP. L'adresse IP du client enregistrée pour une demande transférée peut être falsifiée, elle ne doit donc pas être fiable.

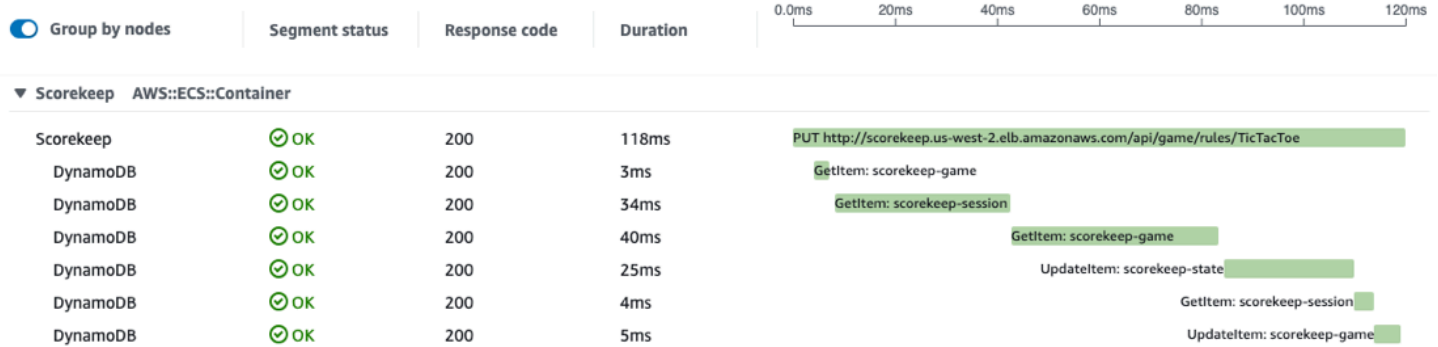
Vous pouvez utiliser le SDK X-Ray pour enregistrer des informations supplémentaires telles que des [annotations et des métadonnées](#). Pour plus d'informations sur la structure et les informations enregistrées dans les segments et les sous-segments, consultez [AWS X-Ray documents segmentés](#). La taille des documents segmentés peut atteindre 64 kB.

Sous-segments

Un segment peut décomposer les données concernant le travail effectué en sous-segments. Les sous-segments fournissent des informations de durée plus précises et des détails sur les appels en aval effectués par votre application pour satisfaire la demande initiale. Un sous-segment peut contenir des informations supplémentaires sur un appel à une Service AWS API HTTP externe

ou à une base de données SQL. Vous pouvez même définir des sous-segments arbitraires pour instrumenter des fonctions spécifiques ou des lignes de code dans votre application.

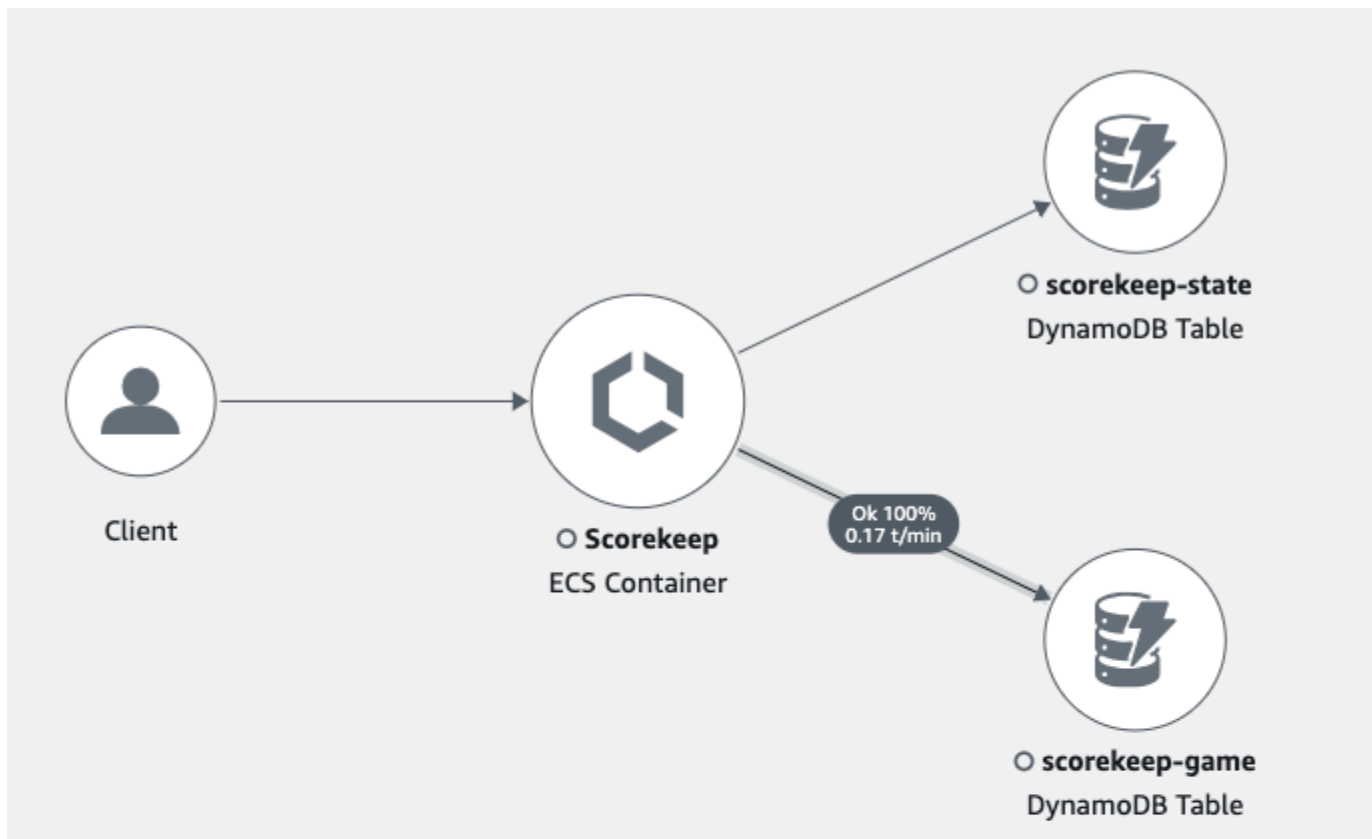
Segments Timeline [Info](#)



Pour les services qui n'envoient pas leurs propres segments, comme Amazon DynamoDB, X-Ray utilise des sous-segments pour générer des segments déduits et des nœuds en aval sur la carte de trace. Cela vous permet de voir l'ensemble de vos dépendances en aval, même si elles ne prennent pas en charge le suivi ou sont externes.

Les sous-segments représentent la vue par votre application d'un appel en aval en tant que client. Si le service en aval est également instrumenté, le segment qu'il envoie remplace le segment déduit généré à partir du sous-segment du client en amont. Le nœud sur le graphique de services utilise toujours des informations provenant du segment du service, si elle sont disponibles, tandis que la zone périphérique entre les deux nœuds utilise le sous-segment du service en amont.

Par exemple, lorsque vous appelez DynamoDB avec un client SDK AWS instrumenté, le SDK X-Ray enregistre un sous-segment pour cet appel. DynamoDB n'envoie pas de segment. Par conséquent, le segment déduit dans le traçage, le nœud DynamoDB sur le graphe de service et la limite entre votre service et DynamoDB contiennent tous des informations provenant du sous-segment.

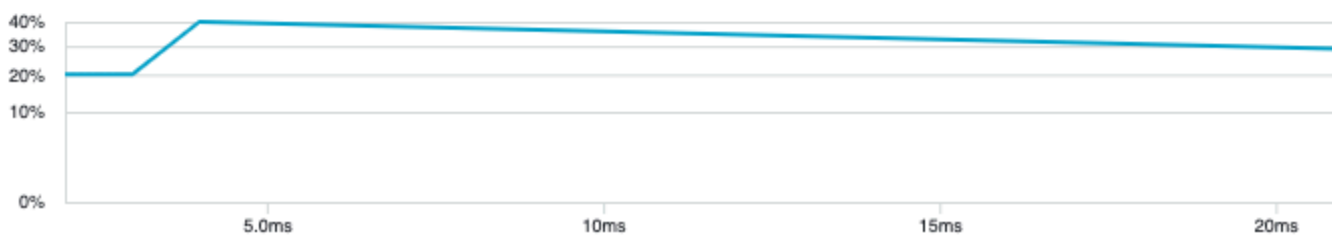


▼ Edge details

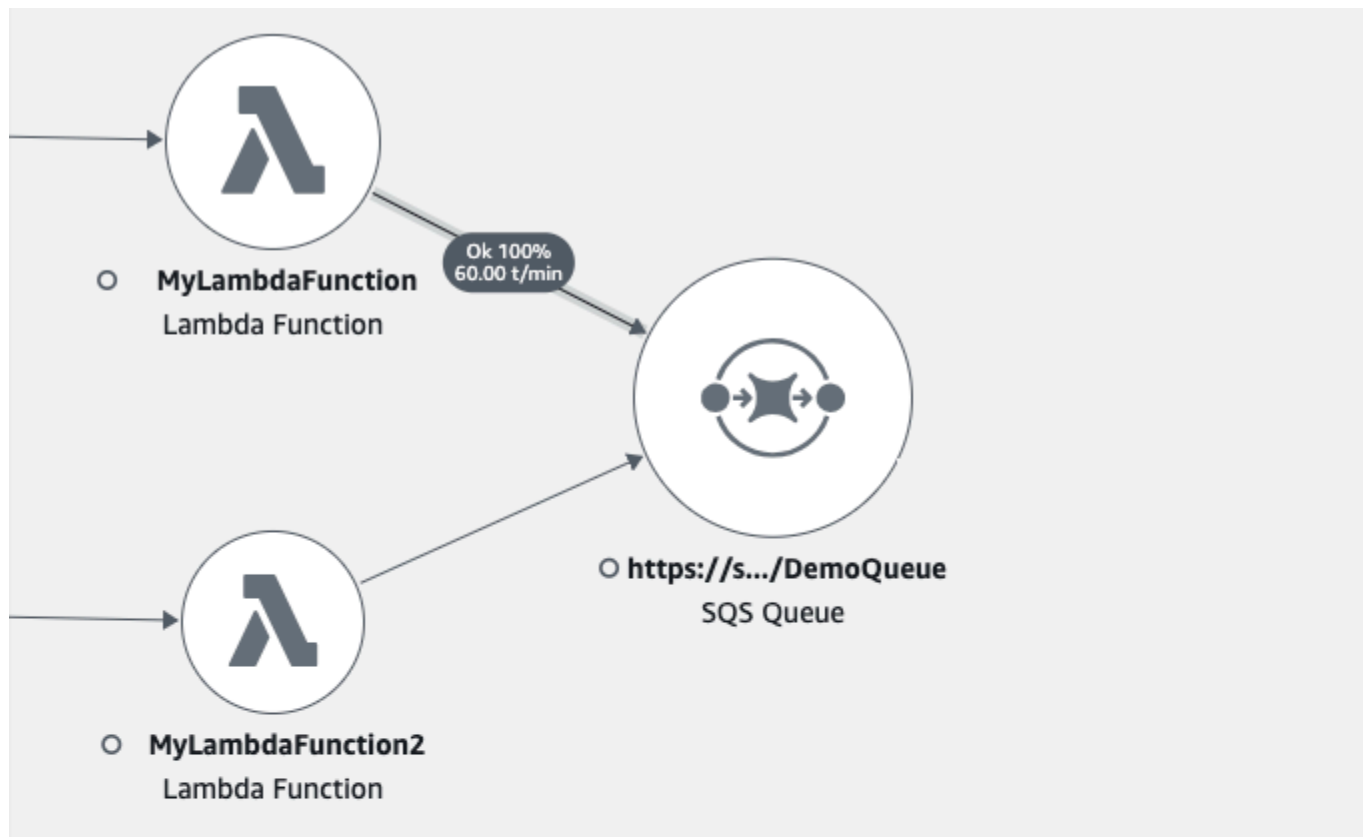
Source: Scorekeep Destination: scorekeep-game

Response time distribution filter

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



Lorsque vous appelez un autre service instrumenté avec une application instrumentée, le service en aval envoie son propre segment pour enregistrer sa vue de l'appel que le service en amont a enregistré dans un sous-segment. Dans le graphique de services, les deux nœuds de service contiennent des informations de synchronisation et des informations d'erreur provenant de ces services, tandis que l'arc entre eux contient des informations provenant du sous-segment du service en amont.



▼ 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.



Ces deux points de vue sont utiles, car le service en aval enregistre précisément le moment auquel il a commencé et terminé le travail sur la demande et le service en amont enregistre la latence de l'aller-retour, y compris le temps passé par la demande pour circuler entre les deux services.

Graphique de services

X-Ray utilise les données envoyées par votre application pour générer un graphe de service. Chaque AWS ressource qui envoie des données à X-Ray apparaît sous forme de service dans le graphique. Les zones périphériques connectent les services qui fonctionnent ensemble pour traiter les demandes. Les arcs connectent les clients à votre application, et votre application aux ressources et services en aval qu'elle utilise.

Noms des services

Un segment name doit correspondre au nom de domaine ou au nom logique du service qui génère le segment. Toutefois, cela n'est pas appliqué. Toute application autorisée [PutTraceSegments](#) peut envoyer des segments sous n'importe quel nom.

Un graphique de services est un document au format JSON qui contient des informations sur les services et ressources qui composent votre application. La console X-Ray utilise le graphe de service pour générer une visualisation ou une carte de service.



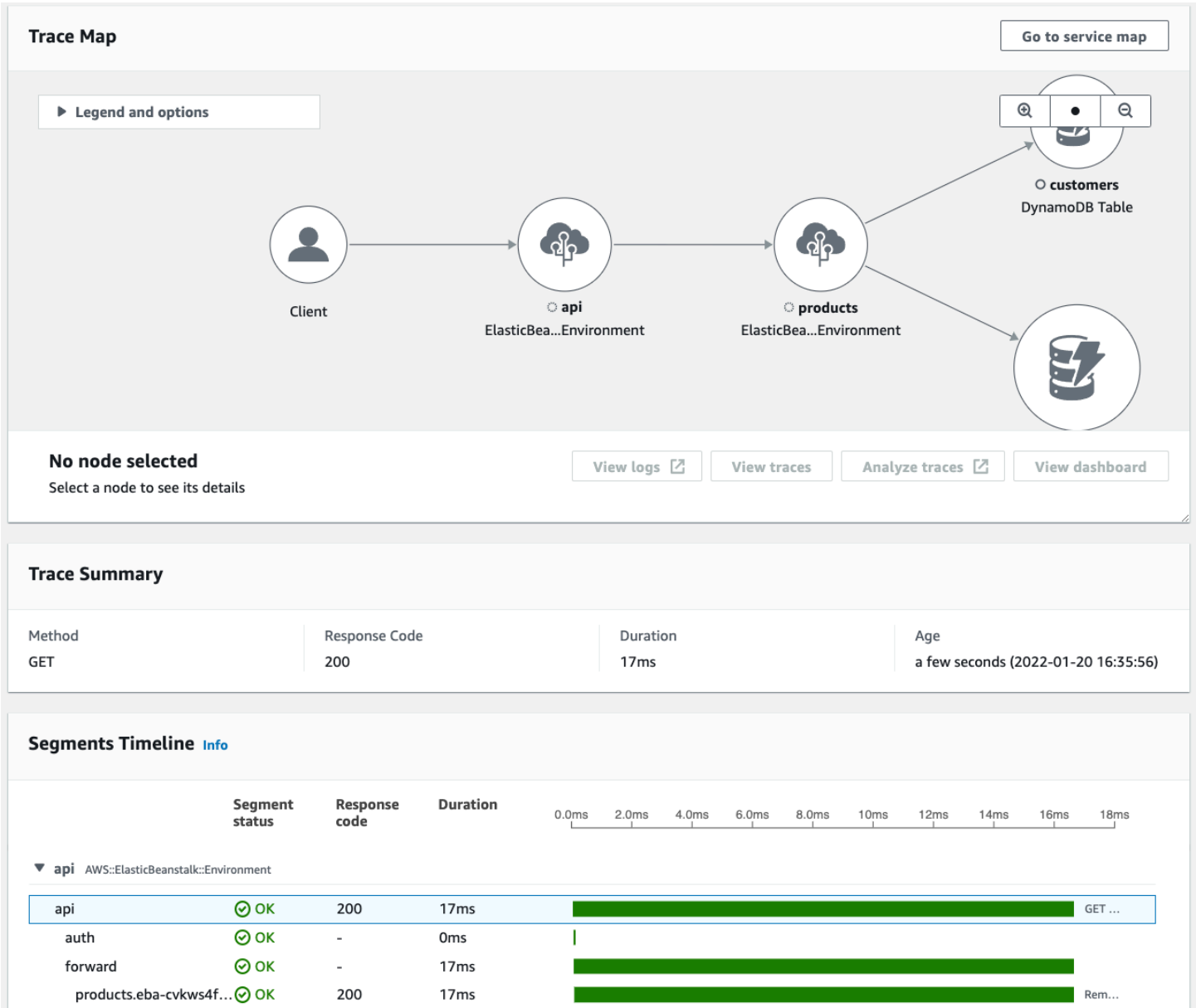
Pour une application distribuée, X-Ray combine les nœuds de tous les services qui traitent les demandes avec le même ID de trace dans un graphe de service unique. Le premier service qui traite la demande ajoute un [en-tête de suivi](#), lequel est ensuite propagé entre le serveur frontal et les services appelés.

Par exemple, [Scorekeep](#) exécute une API web qui appelle un microservice (une fonction AWS Lambda) pour générer un nom aléatoire à l'aide d'une bibliothèque Node.js. Le SDK X-Ray pour Java génère l'ID de trace et l'inclut dans les appels à Lambda. Lambda envoie des données de suivi et transmet l'ID de trace à la fonction. Le SDK X-Ray pour Node.js utilise également l'ID de trace pour envoyer des données. Par conséquent, les nœuds de l'API, du service Lambda et de la fonction Lambda apparaissent tous sous forme de nœuds distincts, mais connectés, sur la carte de trace.

Les données du graphique de service sont conservées pendant 30 jours.

Suivis

Un ID de suivi permet de suivre le cheminement d'une demande au sein de votre application. Un suivi recueille tous les segments générés par une seule demande. Il s'agit généralement d'une requête HTTP GET ou POST qui passe par un équilibreur de charge, accède au code de votre application et génère des appels en aval vers d'autres AWS services ou un site Web APIs externe. Le premier service pris en charge avec lequel la demande HTTP interagit ajoute un en-tête d'ID de suivi à la demande et le propage en aval pour suivre la latence, l'organisation et les autres données de la demande.



Consultez les [AWS X-Ray tarifs](#) pour en savoir plus sur le mode de facturation des X-Ray Traces. Les données de suivi sont conservées pendant 30 jours.

Echantillonnage

Pour garantir un suivi efficace et fournir un échantillon représentatif des demandes traitées par votre application, le SDK X-Ray applique un algorithme d'échantillonnage pour déterminer quelles demandes sont suivies. Par défaut, le SDK X-Ray enregistre la première demande chaque seconde, et 5 % de toutes les demandes supplémentaires.

Afin de ne pas encourir de frais de service lors de la mise en route, le taux d'échantillonnage par défaut est prudent. Vous pouvez configurer X-Ray pour modifier la règle d'échantillonnage par défaut et configurer des règles supplémentaires qui appliquent l'échantillonnage en fonction des propriétés du service ou de la demande.

Par exemple, vous souhaitez peut-être désactiver l'échantillonnage et suivre toutes les demandes d'appels qui modifient l'état ou traitent des utilisateurs ou des transactions. Pour les appels en lecture seule à fort volume, tels que les appels d'interrogation d'arrière-plan, les vérifications de l'état ou la maintenance des connexions, vous pouvez effectuer un échantillon à faible taux et obtenir malgré tout suffisamment de données pour voir tout problème éventuel.

Pour de plus amples informations, veuillez consulter [Configuration de règles d'échantillonnage](#).

En-tête de suivi

Toutes les demandes sont suivies jusqu'à un minimum configurable. Une fois ce minimum atteint, seul un certain pourcentage des demandes est suivi afin d'éviter les coûts superflus. La décision d'échantillonnage et l'ID de suivi sont ajoutés aux demandes HTTP dans des en-têtes de suivi nommés X-Amzn-Trace-Id. Le premier X-Ray-integrated service touché par la demande ajoute un en-tête de suivi, qui est lu par le SDK X-Ray et inclus dans la réponse.

Exemple En-tête de suivi avec ID de suivi racine et décision d'échantillonnage

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

Sécurité de l'en-tête de suivi

Un en-tête de suivi peut provenir du SDK X-Ray Service AWS, d'une demande ou d'une demande du client. Votre application peut supprimer `X-Amzn-Trace-Id` des demandes entrantes afin d'éviter les problèmes causés par le fait que les utilisateurs ajoutent des décisions de suivi IDs ou d'échantillonnage à leurs demandes.

L'en-tête de suivi peut également contenir un ID de segment parent si la demande provenait d'une application instrumentée. Par exemple, si votre application appelle une API Web HTTP en aval avec un client HTTP instrumenté, le SDK X-Ray ajoute l'ID de segment de la demande d'origine à l'en-tête de suivi de la demande en aval. Une application instrumentée qui sert la demande en aval peut enregistrer l'ID de segment parent pour connecter les deux demandes.

Exemple En-tête de suivi avec ID de suivi racine, ID de segment parent et décision d'échantillonnage

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

Lineage peuvent être ajoutés à l'en-tête de trace par Lambda et Services AWS d'autres dans le cadre de leurs mécanismes de traitement, et ne doivent pas être utilisés directement.

Exemple En-tête de traçage avec Lineage

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

Expressions de filtrage

Même avec l'échantillonnage, une application complexe génère beaucoup de données. La AWS X-Ray console fournit une easy-to-navigate vue du graphe de service. Elle affiche des informations de santé et de performances qui vous aide à identifier les problèmes et les occasions d'optimiser votre application. Pour les suivis avancés, vous pouvez explorer les suivis pour les demandes individuelles ou utiliser des expressions de filtrage pour trouver des suivis liés à des chemins ou des utilisateurs spécifiques.

Traces [Info](#)

5m 15m 30m

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

Run query 5 traces retrieved

Query refiners

Traces (5)
This table shows the most recent traces with an average response time of 0.16s. It shows as many as 1000 traces.

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

Groups

En étendant les expressions de filtre, X-Ray prend également en charge la fonctionnalité de groupe. En utilisant une expression de filtrage, vous pouvez définir les critères en vertu desquels vous acceptez les suivis dans le groupe.

Vous pouvez appeler le groupe par son nom ou par son Amazon Resource Name (ARN) pour générer son propre graphe de service, ses résumés de suivi et ses CloudWatch statistiques Amazon. Une fois qu'un groupe est créé, les traces entrantes sont vérifiées par rapport à l'expression du filtre du groupe lorsqu'elles sont stockées dans le service X-Ray. Les mesures relatives au nombre de traces correspondant à chaque critère sont publiées à CloudWatch chaque minute.

Le fait de mettre à jour l'expression de filtrage d'un groupe n'a pas pour effet de modifier les données déjà enregistrées. La mise à jour s'applique uniquement aux suivis ultérieurs. Cela peut entraîner la fusion des expressions nouvelles et anciennes dans le graphique. Pour éviter cela, supprimez le groupe actuel et créez-en un nouveau.

Note

Les groupes sont facturés en fonction du nombre de suivis récupérés qui correspondent à l'expression de filtrage. Pour en savoir plus, consultez [Pricing AWS X-Ray](#) (Tarification).

Pour plus d'informations sur les groupes, consultez [Configuration des groupes](#).

Annotations et métadonnées

Lorsque vous instrumentez votre application, le SDK X-Ray enregistre des informations sur les demandes entrantes et sortantes, les AWS ressources utilisées et l'application elle-même. Vous pouvez ajouter d'autres informations dans le document de segment, comme des annotations et des métadonnées. Les annotations et les métadonnées sont agrégées au niveau du suivi et peuvent être ajoutées à n'importe quel segment ou sous-segment.

Les annotations sont de simples paires clé-valeur, indexées en vue de leur utilisation avec les [expressions de filtre](#). Utilisez les annotations pour enregistrer les données que vous souhaitez utiliser pour regrouper les suivis dans la console ou lors de l'appel de l'API [GetTraceSummaries](#).

X-Ray indexe jusqu'à 50 annotations par trace.

Les métadonnées sont des paires clé-valeur dont les valeurs peuvent être de tout type (objet et liste inclus), mais elles ne sont pas indexées. Utilisez les métadonnées pour enregistrer les données qui doivent être stockées dans le suivi, mais n'ont pas besoin d'être utilisées pour explorer les suivis.

Vous pouvez consulter les annotations et les métadonnées dans la fenêtre des détails du segment ou du sous-segment, sur la page [Détails du suivi](#) de la CloudWatch console.

▼ DynamoDB AWS::DynamoDB::Table					
DynamoDB	✓ OK	200	9ms	GetItem: scorekeep-session	
DynamoDB	✓ OK	200	10ms	UpdateItem: scorekeep-game	
DynamoDB	✓ OK	200	46ms	GetItem: scorekeep-session	
DynamoDB	✓ OK	200	39ms		

Segment details: DynamoDB

Overview	Resources	Annotations	Metadata	Exceptions	SQL

Erreurs, anomalies et exceptions

X-Ray suit les erreurs qui se produisent dans le code de votre application et les erreurs renvoyées par les services en aval. Les erreurs sont classées comme suit :

- **Error**— Erreurs du client (erreurs de la série 400)
- **Fault**— Défaillances du serveur (erreurs de la série 500)
- **Throttle**— Erreurs de régulation (429 demandes de trop)

Lorsqu'une exception se produit alors que votre application traite une demande instrumentée, le SDK X-Ray enregistre les détails de l'exception, y compris le suivi de la pile, le cas échéant. Vous pouvez consulter les exceptions dans [les détails des segments](#) dans la console X-Ray.

Sécurité dans AWS X-Ray

La sécurité du cloud AWS est la priorité absolue. En tant que AWS client, vous bénéficiez d'un centre de données et d'une architecture réseau conçus pour répondre aux exigences des entreprises les plus sensibles en matière de sécurité.

La sécurité est une responsabilité partagée entre vous AWS et vous. Le [modèle de responsabilité partagée](#) décrit cette notion par les termes sécurité du cloud et sécurité dans le cloud :

- **Sécurité du cloud** : AWS est chargée de protéger l'infrastructure qui s'exécute Services AWS dans le AWS Cloud. AWS vous fournit également des services que vous pouvez utiliser en toute sécurité. L'efficacité de notre sécurité est régulièrement testée et vérifiée par des auditeurs tiers dans le cadre des [programmes de conformité AWS](#). Pour en savoir plus sur les programmes de conformité qui s'appliquent à X-Ray, consultez Services AWS la section [Scope by Compliance Program](#).
- **Sécurité dans le cloud** — Votre responsabilité est déterminée par Service AWS ce que vous utilisez. Vous êtes également responsable d'autres facteurs, y compris la sensibilité de vos données, les exigences de votre organisation ainsi que les lois et réglementations applicables.

Cette documentation vous aidera à comprendre comment appliquer le modèle de responsabilité partagée lors de l'utilisation de X-Ray. Les rubriques suivantes expliquent comment configurer X-Ray pour atteindre vos objectifs de sécurité et de conformité. Vous apprendrez également à utiliser d'autres outils Services AWS qui peuvent vous aider à surveiller et à sécuriser vos ressources X-Ray.

Rubriques

- [Protection des données dans AWS X-Ray](#)
- [Gestion des identités et des accès pour AWS X-Ray](#)
- [Validation de conformité pour AWS X-Ray](#)
- [Résilience dans AWS X-Ray](#)
- [Sécurité de l'infrastructure dans AWS X-Ray](#)

Protection des données dans AWS X-Ray

AWS X-Ray chiffre toujours les traces et les données associées au repos. Lorsque vous devez auditer et désactiver des clés de chiffrement pour des raisons de conformité ou pour des exigences

internes, vous pouvez configurer X-Ray pour qu'il utilise une clé AWS Key Management Service (AWS KMS) pour chiffrer les données.

X-Ray fournit un Clé gérée par AWS nomaws/xray. Utilisez cette clé si vous voulez uniquement effectuer un [audit de l'utilisation des clés dans AWS CloudTrail](#) et que vous n'avez pas besoin de gérer la clé elle-même. Lorsque vous devez gérer l'accès à la clé ou configurer la rotation des clés, vous pouvez [créer une clé gérée par le client](#).

Lorsque vous modifiez les paramètres de chiffrement, X-Ray passe un certain temps à générer et à propager des clés de données. Pendant que la nouvelle clé est en cours de traitement, X-Ray peut chiffrer les données avec une combinaison de paramètres nouveaux et anciens. Les données existantes ne sont pas chiffrées à nouveau lorsque vous modifiez les paramètres de chiffrement.

 Note

AWS KMS se facture lorsque X-Ray utilise une clé KMS pour chiffrer ou déchiffrer les données de suivi.

- Chiffrement par défaut — Gratuit.
- Clé gérée par AWS— Payez pour l'utilisation des clés.
- clé gérée par le client — Payez pour le stockage et l'utilisation des clés.

Consultez les [AWS Key Management Service tarifs](#) pour plus de détails.

 Note

Les notifications X-Ray Insights envoient des événements à Amazon EventBridge, qui ne prend actuellement pas en charge les clés gérées par les clients. Pour plus d'informations, consultez la section [Protection des données sur Amazon EventBridge](#).

Vous devez disposer d'un accès de niveau utilisateur à une clé gérée par le client pour configurer X-Ray afin de l'utiliser, puis pour afficher les traces chiffrées. Pour plus d'informations, consultez [Autorisations utilisateur pour le chiffrement](#).

CloudWatch console

Pour configurer X-Ray afin d'utiliser une clé KMS pour le chiffrement à l'aide de la CloudWatch console

1. Connectez-vous à la CloudWatch console AWS Management Console et ouvrez-la à l'adresse <https://console.aws.amazon.com/cloudwatch/>.
2. Choisissez Paramètres dans le volet de navigation de gauche.
3. Choisissez Afficher les paramètres sous Encryption dans la section X-Ray Traces.
4. Choisissez Modifier dans la section Configuration du chiffrement.
5. Choisissez Utiliser une clé KMS.
6. Choisissez une clé dans le menu déroulant :
 - aws/xray — Utilisez le. Clé gérée par AWS
 - alias de clé — Utilisez une clé gérée par le client dans votre compte.
 - Entrez manuellement un ARN de clé : utilisez une clé gérée par le client dans un autre compte. Saisissez l'Amazon Resource Name (ARN) complet de la clé dans le champ qui s'affiche.
7. Choisissez Mettre à jour le chiffrement.

X-Ray console

Pour configurer X-Ray afin d'utiliser une clé KMS pour le chiffrement à l'aide de la console X-Ray

1. Ouvrez la [console X-Ray](#).
2. Choisissez Chiffrement.
3. Choisissez Utiliser une clé KMS.
4. Choisissez une clé dans le menu déroulant :
 - aws/xray — Utilisez le. Clé gérée par AWS
 - alias de clé — Utilisez une clé gérée par le client dans votre compte.
 - Entrez manuellement un ARN de clé : utilisez une clé gérée par le client dans un autre compte. Saisissez l'Amazon Resource Name (ARN) complet de la clé dans le champ qui s'affiche.
5. Choisissez Appliquer.

 Note

X-Ray ne prend pas en charge les clés KMS asymétriques.

Si X-Ray ne parvient pas à accéder à votre clé de chiffrement, il arrête de stocker les données. Cela peut se produire si votre utilisateur perd l'accès à la clé KMS ou si vous désactivez une clé actuellement utilisée. Lorsque cela se produit, X-Ray affiche une notification dans la barre de navigation.

Pour configurer les paramètres de chiffrement avec l'API X-Ray, consultez [Configuration de l'échantillonnage, des groupes et des paramètres de chiffrement avec l'API AWS X-Ray](#).

Gestion des identités et des accès pour AWS X-Ray

AWS Identity and Access Management (IAM) est un outil Service AWS qui permet à un administrateur de contrôler en toute sécurité l'accès aux AWS ressources. Les administrateurs IAM contrôlent qui peut être authentifié (connecté) et autorisé (autorisé) à utiliser les ressources X-Ray. IAM est un Service AWS outil que vous pouvez utiliser sans frais supplémentaires.

Rubriques

- [Public ciblé](#)
- [Authentification par des identités](#)
- [Gestion des accès à l'aide de politiques](#)
- [Comment AWS X-Ray fonctionne avec IAM](#)
- [AWS X-Ray exemples de politiques basées sur l'identité](#)
- [Résolution des problèmes AWS X-Ray d'identité et d'accès](#)

Public ciblé

La façon dont vous utilisez AWS Identity and Access Management (IAM) varie en fonction du travail que vous effectuez dans X-Ray.

Utilisateur du service : si vous utilisez le service X-Ray pour effectuer votre travail, votre administrateur vous fournit les informations d'identification et les autorisations dont vous avez besoin. Au fur et à mesure que vous utilisez de plus en plus de fonctionnalités de X-Ray pour effectuer votre

travail, vous aurez peut-être besoin d'autorisations supplémentaires. En comprenant bien la gestion des accès, vous saurez demander les autorisations appropriées à votre administrateur. Si vous ne parvenez pas à accéder à une fonctionnalité de X-Ray, consultez [Résolution des problèmes AWS X-Ray d'identité et d'accès](#).

Administrateur du service — Si vous êtes responsable des ressources X-Ray de votre entreprise, vous avez probablement un accès complet à X-Ray. C'est à vous de déterminer les fonctionnalités et les ressources de X-Ray auxquelles les utilisateurs de votre service doivent accéder. Vous devez ensuite soumettre les demandes à votre administrateur IAM pour modifier les autorisations des utilisateurs de votre service. Consultez les informations sur cette page pour comprendre les concepts de base d'IAM. Pour en savoir plus sur la façon dont votre entreprise peut utiliser IAM avec X-Ray, consultez [Comment AWS X-Ray fonctionne avec IAM](#).

Administrateur IAM — Si vous êtes administrateur IAM, vous souhaitez peut-être en savoir plus sur la manière dont vous pouvez rédiger des politiques pour gérer l'accès à X-Ray. Pour consulter des exemples de politiques basées sur l'identité de X-Ray que vous pouvez utiliser dans IAM, consultez [AWS X-Ray exemples de politiques basées sur l'identité](#)

Authentification par des identités

L'authentification est la façon dont vous vous connectez à AWS l'aide de vos informations d'identification. Vous devez être authentifié (connecté à AWS) en tant qu'utilisateur IAM ou en assumant un rôle IAM. Utilisateur racine d'un compte AWS

Vous pouvez vous connecter en AWS tant qu'identité fédérée en utilisant les informations d'identification fournies par le biais d'une source d'identité. AWS IAM Identity Center Les utilisateurs (IAM Identity Center), l'authentification unique de votre entreprise et vos informations d'identification Google ou Facebook sont des exemples d'identités fédérées. Lorsque vous vous connectez avec une identité fédérée, votre administrateur aura précédemment configuré une fédération d'identités avec des rôles IAM. Lorsque vous accédez à AWS l'aide de la fédération, vous assumez indirectement un rôle.

Selon le type d'utilisateur que vous êtes, vous pouvez vous connecter au portail AWS Management Console ou au portail AWS d'accès. Pour plus d'informations sur la connexion à AWS, consultez la section [Comment vous connecter à votre compte Compte AWS dans](#) le guide de Connexion à AWS l'utilisateur.

Si vous y accédez AWS par programmation, AWS fournit un kit de développement logiciel (SDK) et une interface de ligne de commande (CLI) pour signer cryptographiquement vos demandes à l'aide

de vos informations d'identification. Si vous n'utilisez pas d' AWS outils, vous devez signer vous-même les demandes. Pour plus d'informations sur l'utilisation de la méthode recommandée pour signer des demandes vous-même, consultez [AWS Signature Version 4 pour les demandes d'API](#) dans le Guide de l'utilisateur IAM.

Quelle que soit la méthode d'authentification que vous utilisez, vous devrez peut-être fournir des informations de sécurité supplémentaires. Par exemple, il vous AWS recommande d'utiliser l'authentification multifactorielle (MFA) pour renforcer la sécurité de votre compte. Pour plus d'informations, consultez [Authentification multifactorielle](#) dans le Guide de l'utilisateur AWS IAM Identity Center et [Authentification multifactorielle AWS dans IAM](#) dans le Guide de l'utilisateur IAM.

Compte AWS utilisateur root

Lorsque vous créez un Compte AWS, vous commencez par une identité de connexion unique qui donne un accès complet à toutes Services AWS les ressources du compte. Cette identité est appelée utilisateur Compte AWS root et est accessible en vous connectant avec l'adresse e-mail et le mot de passe que vous avez utilisés pour créer le compte. Il est vivement recommandé de ne pas utiliser l'utilisateur racine pour vos tâches quotidiennes. Protégez vos informations d'identification d'utilisateur racine et utilisez-les pour effectuer les tâches que seul l'utilisateur racine peut effectuer. Pour obtenir la liste complète des tâches qui vous imposent de vous connecter en tant qu'utilisateur racine, consultez [Tâches nécessitant les informations d'identification de l'utilisateur racine](#) dans le Guide de l'utilisateur IAM.

Utilisateurs et groupes IAM

Un [utilisateur IAM](#) est une identité au sein de vous Compte AWS qui possède des autorisations spécifiques pour une seule personne ou une seule application. Dans la mesure du possible, nous vous recommandons de vous appuyer sur des informations d'identification temporaires plutôt que de créer des utilisateurs IAM ayant des informations d'identification à long terme telles que des mots de passe et des clés d'accès. Toutefois, si certains cas d'utilisation spécifiques nécessitent des informations d'identification à long terme avec les utilisateurs IAM, nous vous recommandons d'effectuer une rotation des clés d'accès. Pour plus d'informations, consultez [Rotation régulière des clés d'accès pour les cas d'utilisation nécessitant des informations d'identification](#) dans le Guide de l'utilisateur IAM.

Un [groupe IAM](#) est une identité qui concerne un ensemble d'utilisateurs IAM. Vous ne pouvez pas vous connecter en tant que groupe. Vous pouvez utiliser les groupes pour spécifier des autorisations pour plusieurs utilisateurs à la fois. Les groupes permettent de gérer plus facilement les autorisations

pour de grands ensembles d'utilisateurs. Par exemple, vous pouvez nommer un groupe IAMAdminset lui donner les autorisations nécessaires pour administrer les ressources IAM.

Les utilisateurs sont différents des rôles. Un utilisateur est associé de manière unique à une personne ou une application, alors qu'un rôle est conçu pour être endossé par tout utilisateur qui en a besoin. Les utilisateurs disposent d'informations d'identification permanentes, mais les rôles fournissent des informations d'identification temporaires. Pour plus d'informations, consultez [Cas d'utilisation pour les utilisateurs IAM](#) dans le Guide de l'utilisateur IAM.

Rôles IAM

Un [rôle IAM](#) est une identité au sein de votre Compte AWS dotée d'autorisations spécifiques. Le concept ressemble à celui d'utilisateur IAM, mais le rôle IAM n'est pas associé à une personne en particulier. Pour assumer temporairement un rôle IAM dans le AWS Management Console, vous pouvez [passer d'un rôle d'utilisateur à un rôle IAM \(console\)](#). Vous pouvez assumer un rôle en appelant une opération d' AWS API AWS CLI ou en utilisant une URL personnalisée. Pour plus d'informations sur les méthodes d'utilisation des rôles, consultez [Méthodes pour endosser un rôle](#) dans le Guide de l'utilisateur IAM.

Les rôles IAM avec des informations d'identification temporaires sont utiles dans les cas suivants :

- Accès utilisateur fédéré : pour attribuer des autorisations à une identité fédérée, vous créez un rôle et définissez des autorisations pour le rôle. Quand une identité externe s'authentifie, l'identité est associée au rôle et reçoit les autorisations qui sont définies par celui-ci. Pour obtenir des informations sur les rôles pour la fédération, consultez [Création d'un rôle pour un fournisseur d'identité tiers \(fédération\)](#) dans le Guide de l'utilisateur IAM. Si vous utilisez IAM Identity Center, vous configurez un jeu d'autorisations. IAM Identity Center met en corrélation le jeu d'autorisations avec un rôle dans IAM afin de contrôler à quoi vos identités peuvent accéder après leur authentification. Pour plus d'informations sur les jeux d'autorisations, consultez [Jeux d'autorisations](#) dans le Guide de l'utilisateur AWS IAM Identity Center .
- Autorisations d'utilisateur IAM temporaires : un rôle ou un utilisateur IAM peut endosser un rôle IAM pour profiter temporairement d'autorisations différentes pour une tâche spécifique.
- Accès intercompte : vous pouvez utiliser un rôle IAM pour permettre à un utilisateur (principal de confiance) d'un compte différent d'accéder aux ressources de votre compte. Les rôles constituent le principal moyen d'accorder l'accès intercompte. Toutefois, dans certains Services AWS cas, vous pouvez associer une politique directement à une ressource (au lieu d'utiliser un rôle comme proxy). Pour en savoir plus sur la différence entre les rôles et les politiques basées sur les

ressources pour l'accès intercompte, consultez [Accès intercompte aux ressources dans IAM](#) dans le Guide de l'utilisateur IAM.

- Accès multiservices — Certains Services AWS utilisent des fonctionnalités dans d'autres Services AWS. Par exemple, lorsque vous effectuez un appel dans un service, il est courant que ce service exécute des applications dans Amazon EC2 ou stocke des objets dans Amazon S3. Un service peut le faire en utilisant les autorisations d'appel du principal, un rôle de service ou un rôle lié au service.
- Sessions d'accès direct (FAS) : lorsque vous utilisez un utilisateur ou un rôle IAM pour effectuer des actions AWS, vous êtes considéré comme un mandant. Lorsque vous utilisez certains services, vous pouvez effectuer une action qui initie une autre action dans un autre service. FAS utilise les autorisations du principal appelant et Service AWS, associées Service AWS à la demande, pour adresser des demandes aux services en aval. Les demandes FAS ne sont effectuées que lorsqu'un service reçoit une demande qui nécessite des interactions avec d'autres personnes Services AWS ou des ressources pour être traitée. Dans ce cas, vous devez disposer d'autorisations nécessaires pour effectuer les deux actions. Pour plus de détails sur une politique lors de la formulation de demandes FAS, consultez [Transmission des sessions d'accès](#).
- Rôle de service : il s'agit d'un [rôle IAM](#) attribué à un service afin de réaliser des actions en votre nom. Un administrateur IAM peut créer, modifier et supprimer un rôle de service à partir d'IAM. Pour plus d'informations, consultez [Création d'un rôle pour la délégation d'autorisations à un Service AWS](#) dans le Guide de l'utilisateur IAM.
- Rôle lié à un service — Un rôle lié à un service est un type de rôle de service lié à un. Service AWS Le service peut endosser le rôle afin d'effectuer une action en votre nom. Les rôles liés à un service apparaissent dans votre Compte AWS répertoire et appartiennent au service. Un administrateur IAM peut consulter, mais ne peut pas modifier, les autorisations concernant les rôles liés à un service.
- Applications exécutées sur Amazon EC2 : vous pouvez utiliser un rôle IAM pour gérer les informations d'identification temporaires pour les applications qui s'exécutent sur une EC2 instance et qui envoient des demandes AWS CLI d' AWS API. Cela est préférable au stockage des clés d'accès dans l' EC2 instance. Pour attribuer un AWS rôle à une EC2 instance et le rendre disponible pour toutes ses applications, vous devez créer un profil d'instance attaché à l'instance. Un profil d'instance contient le rôle et permet aux programmes exécutés sur l' EC2 instance d'obtenir des informations d'identification temporaires. Pour plus d'informations, consultez [Utiliser un rôle IAM pour accorder des autorisations aux applications exécutées sur des EC2 instances Amazon](#) dans le guide de l'utilisateur IAM.

Gestion des accès à l'aide de politiques

Vous contrôlez l'accès en AWS créant des politiques et en les associant à AWS des identités ou à des ressources. Une politique est un objet AWS qui, lorsqu'il est associé à une identité ou à une ressource, définit leurs autorisations. AWS évalue ces politiques lorsqu'un principal (utilisateur, utilisateur root ou session de rôle) fait une demande. Les autorisations dans les politiques déterminent si la demande est autorisée ou refusée. La plupart des politiques sont stockées AWS sous forme de documents JSON. Pour plus d'informations sur la structure et le contenu des documents de politique JSON, consultez [Vue d'ensemble des politiques JSON](#) dans le Guide de l'utilisateur IAM.

Les administrateurs peuvent utiliser les politiques AWS JSON pour spécifier qui a accès à quoi. C'est-à-dire, quel principal peut effectuer des actions sur quelles ressources et dans quelles conditions.

Par défaut, les utilisateurs et les rôles ne disposent d'aucune autorisation. Pour octroyer aux utilisateurs des autorisations d'effectuer des actions sur les ressources dont ils ont besoin, un administrateur IAM peut créer des politiques IAM. L'administrateur peut ensuite ajouter les politiques IAM aux rôles et les utilisateurs peuvent assumer les rôles.

Les politiques IAM définissent les autorisations d'une action, quelle que soit la méthode que vous utilisez pour exécuter l'opération. Par exemple, supposons que vous disposiez d'une politique qui autorise l'action `iam:GetRole`. Un utilisateur appliquant cette politique peut obtenir des informations sur le rôle à partir de AWS Management Console AWS CLI, de ou de l' AWS API.

Politiques basées sur l'identité

Les politiques basées sur l'identité sont des documents de politique d'autorisations JSON que vous pouvez attacher à une identité telle qu'un utilisateur, un groupe d'utilisateurs ou un rôle IAM. Ces politiques contrôlent quel type d'actions des utilisateurs et des rôles peuvent exécuter, sur quelles ressources et dans quelles conditions. Pour découvrir comment créer une politique basée sur l'identité, consultez [Définition d'autorisations IAM personnalisées avec des politiques gérées par le client](#) dans le Guide de l'utilisateur IAM.

Les politiques basées sur l'identité peuvent être classées comme des politiques en ligne ou des politiques gérées. Les politiques en ligne sont intégrées directement à un utilisateur, groupe ou rôle. Les politiques gérées sont des politiques autonomes que vous pouvez associer à plusieurs utilisateurs, groupes et rôles au sein de votre Compte AWS. Les politiques gérées incluent les politiques AWS gérées et les politiques gérées par le client. Pour découvrir comment choisir entre

une politique gérée et une politique en ligne, consultez [Choix entre les politiques gérées et les politiques en ligne](#) dans le Guide de l'utilisateur IAM.

Politiques basées sur les ressources

Les politiques basées sur les ressources sont des documents de politique JSON que vous attachez à une ressource. Par exemple, les politiques de confiance de rôle IAM et les politiques de compartiment Amazon S3 sont des politiques basées sur les ressources. Dans les services qui sont compatibles avec les politiques basées sur les ressources, les administrateurs de service peuvent les utiliser pour contrôler l'accès à une ressource spécifique. Pour la ressource dans laquelle se trouve la politique, cette dernière définit quel type d'actions un principal spécifié peut effectuer sur cette ressource et dans quelles conditions. Vous devez [spécifier un principal](#) dans une politique basée sur les ressources. Les principaux peuvent inclure des comptes, des utilisateurs, des rôles, des utilisateurs fédérés ou. Services AWS

Les politiques basées sur les ressources sont des politiques en ligne situées dans ce service. Vous ne pouvez pas utiliser les politiques AWS gérées par IAM dans une stratégie basée sur les ressources.

Listes de contrôle d'accès (ACLs)

Les listes de contrôle d'accès (ACLs) contrôlent les principaux (membres du compte, utilisateurs ou rôles) autorisés à accéder à une ressource. ACLs sont similaires aux politiques basées sur les ressources, bien qu'elles n'utilisent pas le format de document de politique JSON.

Amazon S3 et AWS WAF Amazon VPC sont des exemples de services compatibles. ACLs Pour en savoir plus ACLs, consultez la [présentation de la liste de contrôle d'accès \(ACL\)](#) dans le guide du développeur Amazon Simple Storage Service.

Autres types de politique

AWS prend en charge d'autres types de politiques moins courants. Ces types de politiques peuvent définir le nombre maximum d'autorisations qui vous sont accordées par des types de politiques plus courants.

- **Limite d'autorisations** : une limite d'autorisations est une fonctionnalité avancée dans laquelle vous définissez le nombre maximal d'autorisations qu'une politique basée sur l'identité peut accorder à une entité IAM (utilisateur ou rôle IAM). Vous pouvez définir une limite d'autorisations pour une entité. Les autorisations en résultant représentent la combinaison des politiques basées sur

l'identité d'une entité et de ses limites d'autorisation. Les politiques basées sur les ressources qui spécifient l'utilisateur ou le rôle dans le champ `Principal` ne sont pas limitées par les limites d'autorisations. Un refus explicite dans l'une de ces politiques annule l'autorisation. Pour plus d'informations sur les limites d'autorisations, consultez [Limites d'autorisations pour des entités IAM](#) dans le Guide de l'utilisateur IAM.

- **Politiques de contrôle des services (SCPs)** : SCPs politiques JSON qui spécifient les autorisations maximales pour une organisation ou une unité organisationnelle (UO) dans AWS Organizations. AWS Organizations est un service permettant de regrouper et de gérer de manière centralisée Comptes AWS les multiples propriétés de votre entreprise. Si vous activez toutes les fonctionnalités d'une organisation, vous pouvez appliquer des politiques de contrôle des services (SCPs) à l'un ou à l'ensemble de vos comptes. Le SCP limite les autorisations pour les entités figurant dans les comptes des membres, y compris chacune Utilisateur racine d'un compte AWS d'entre elles. Pour plus d'informations sur les Organizations et consultez SCPs les [politiques de contrôle des services](#) dans le Guide de AWS Organizations l'utilisateur.
- **Politiques de contrôle des ressources (RCPs)** : RCPs politiques JSON que vous pouvez utiliser pour définir le maximum d'autorisations disponibles pour les ressources de vos comptes sans mettre à jour les politiques IAM associées à chaque ressource que vous possédez. Le RCP limite les autorisations pour les ressources des comptes membres et peut avoir un impact sur les autorisations effectives pour les identités, y compris Utilisateur racine d'un compte AWS, qu'elles appartiennent ou non à votre organisation. Pour plus d'informations sur les Organizations RCPs, y compris une liste de ces Services AWS supports RCPs, voir [Politiques de contrôle des ressources \(RCPs\)](#) dans le guide de AWS Organizations l'utilisateur.
- **Politiques de séance** : les politiques de séance sont des politiques avancées que vous utilisez en tant que paramètre lorsque vous créez par programmation une séance temporaire pour un rôle ou un utilisateur fédéré. Les autorisations de séance en résultant sont une combinaison des politiques basées sur l'identité de l'utilisateur ou du rôle et des politiques de séance. Les autorisations peuvent également provenir d'une politique basée sur les ressources. Un refus explicite dans l'une de ces politiques annule l'autorisation. Pour plus d'informations, consultez [Politiques de session](#) dans le Guide de l'utilisateur IAM.

Plusieurs types de politique

Lorsque plusieurs types de politiques s'appliquent à la requête, les autorisations en résultant sont plus compliquées à comprendre. Pour savoir comment AWS déterminer s'il faut autoriser une demande lorsque plusieurs types de politiques sont impliqués, consultez la section [Logique d'évaluation des politiques](#) dans le guide de l'utilisateur IAM.

Comment AWS X-Ray fonctionne avec IAM

Avant d'utiliser IAM pour gérer l'accès à X-Ray, vous devez connaître les fonctionnalités IAM disponibles avec X-Ray. Pour obtenir une vue d'ensemble de la façon dont X-Ray et d'autres appareils Services AWS fonctionnent avec IAM, voir Services AWS That [Work with IAM](#) dans le guide de l'utilisateur d'IAM.

Vous pouvez utiliser AWS Identity and Access Management (IAM) pour accorder des autorisations X-Ray aux utilisateurs et aux ressources de calcul de votre compte. IAM contrôle l'accès au service X-Ray au niveau de l'API afin d'appliquer les autorisations de manière uniforme, quel que soit le client (console, AWS SDK AWS CLI) utilisé par vos utilisateurs.

Pour [utiliser la console X-Ray](#) afin de visualiser des cartes de traçage et des segments, vous n'avez besoin que d'autorisations de lecture. Pour activer l'accès à la console, ajoutez la `AWSXrayReadOnlyAccess` [politique gérée](#) à votre utilisateur IAM.

Pour [le développement et les tests locaux](#), créez un rôle IAM avec des autorisations de lecture et d'écriture. [Assumez le rôle](#) et stockez les informations d'identification temporaires pour le rôle. Vous pouvez utiliser ces informations d'identification avec le daemon X-Ray AWS CLI, le et le AWS SDK. Pour plus d'informations, reportez-vous à la section [Utilisation d'informations d'identification AWS CLI de sécurité temporaires avec le](#).

Pour [déployer votre application instrumentée sur AWS](#), créez un rôle IAM avec des autorisations d'écriture et attribuez-le aux ressources qui exécutent votre application. `AWSXRayDaemonWriteAccess` inclut l'autorisation de télécharger des traces, ainsi que certaines autorisations de lecture pour soutenir l'utilisation de [règles d'échantillonnage](#).

Les stratégies de lecture et d'écriture n'incluent pas l'autorisation de configurer les [paramètres de la clé de chiffrement](#) et les règles d'échantillonnage. `AWSXrayFullAccess` Utilisez-le pour accéder à ces paramètres ou ajouter une [configuration APIs](#) dans une politique personnalisée. Pour les opérations de chiffrement et de déchiffrement avec une clé gérée par le client que vous créez, vous avez également besoin de l'[autorisation d'utiliser la clé](#).

Rubriques

- [Politiques basées sur l'identité de X-Ray](#)
- [Politiques basées sur les ressources de X-Ray](#)
- [Autorisation basée sur les tags X-Ray](#)
- [Exécution de votre application en local](#)

- [Exécution de votre application dans AWS](#)
- [Autorisations utilisateur pour le chiffrement](#)

Politiques basées sur l'identité de X-Ray

Avec les politiques IAM basées sur l'identité, vous pouvez spécifier des actions et ressources autorisées ou refusées, ainsi que les conditions dans lesquelles les actions sont autorisées ou refusées. X-Ray prend en charge des actions, des ressources et des clés de condition spécifiques. Pour en savoir plus sur tous les éléments que vous utilisez dans une politique JSON, consultez [Références des éléments de politique JSON IAM](#) dans le Guide de l'utilisateur IAM.

Actions

Les administrateurs peuvent utiliser les politiques AWS JSON pour spécifier qui a accès à quoi. C'est-à-dire, quel principal peut effectuer des actions sur quelles ressources et dans quelles conditions.

L'élément `Action` d'une politique JSON décrit les actions que vous pouvez utiliser pour autoriser ou refuser l'accès à une politique. Les actions de stratégie portent généralement le même nom que l'opération AWS d'API associée. Il existe quelques exceptions, telles que les actions avec autorisations uniquement qui n'ont pas d'opération API correspondante. Certaines opérations nécessitent également plusieurs actions dans une politique. Ces actions supplémentaires sont nommées actions dépendantes.

Intégration d'actions dans une politique afin d'accorder l'autorisation d'exécuter les opérations associées.

Les actions politiques dans X-Ray utilisent le préfixe suivant avant l'action : `xray:`. Par exemple, pour autoriser quelqu'un à récupérer les détails des ressources d'un groupe à l'aide de l'opération de l'GetGroupAPI X-Ray, vous devez inclure `xray:GetGroupaction` dans sa politique. Les déclarations de politique doivent inclure un élément `Action` ou `NotAction`. X-Ray définit son propre ensemble d'actions décrivant les tâches que vous pouvez effectuer avec ce service.

Pour spécifier plusieurs actions dans une seule déclaration, séparez-les par des virgules comme suit :

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

```
"xray:action2"
```

Vous pouvez aussi spécifier plusieurs actions à l'aide de caractères génériques (*). Par exemple, pour spécifier toutes les actions qui commencent par le mot `Get`, incluez l'action suivante :

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

Pour consulter la liste des actions X-Ray, reportez-vous à la section [Actions définies par AWS X-Ray](#) dans le guide de l'utilisateur IAM.

Ressources

Les administrateurs peuvent utiliser les politiques AWS JSON pour spécifier qui a accès à quoi. C'est-à-dire, quel principal peut effectuer des actions sur quelles ressources et dans quelles conditions.

L'élément de politique JSON `Resource` indique le ou les objets auxquels l'action s'applique. Les instructions doivent inclure un élément `Resource` ou `NotResource`. Il est recommandé de définir une ressource à l'aide de son [Amazon Resource Name \(ARN\)](#). Vous pouvez le faire pour des actions qui prennent en charge un type de ressource spécifique, connu sous la dénomination autorisations de niveau ressource.

Pour les actions qui ne sont pas compatibles avec les autorisations de niveau ressource, telles que les opérations de liste, utilisez un caractère générique (*) afin d'indiquer que l'instruction s'applique à toutes les ressources.

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

Vous pouvez contrôler l'accès aux ressources à l'aide d'une politique IAM. Pour les actions qui prennent en charge les autorisations au niveau des ressources, vous utilisez un Amazon Resource Name (ARN) pour identifier la ressource à laquelle la stratégie s'applique.

Toutes les actions X-Ray peuvent être utilisées dans une politique IAM pour accorder ou refuser aux utilisateurs l'autorisation d'utiliser cette action. Cependant, les [actions X-Ray](#) ne prennent pas toutes en charge les autorisations au niveau des ressources, qui vous permettent de spécifier les ressources sur lesquelles une action peut être effectuée.

Pour les actions qui ne prennent pas en charge les autorisations de niveau ressource, vous devez utiliser « * » comme ressource.

Les actions X-Ray suivantes prennent en charge les autorisations au niveau des ressources :

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

Vous trouverez ci-dessous un exemple de stratégie d'autorisations basées sur l'identité pour une action CreateGroup. Cet exemple montre comment se servir d'un ARN lié à un nom de groupe local-users, en utilisant l'ID unique en tant que caractère générique. L'ID unique est généré lorsque le groupe est créé. Il ne peut donc pas être prédit à l'avance dans la stratégie. Lorsque vous utilisez GetGroup, UpdateGroup ou DeleteGroup, vous pouvez définir l'ID unique en tant que caractère générique ou ARN exact, comprenant l'ID.

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

Vous trouverez ci-dessous un exemple de stratégie d'autorisations basées sur l'identité pour une action CreateSamplingRule.

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

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

Note

L'ARN d'une règle d'échantillonnage est défini par le nom de cette dernière. Contrairement aux règles de groupe ARNs, les règles d'échantillonnage n'ont pas d'identifiant généré de manière unique.

Pour consulter la liste des types de ressources X-Ray et leurs caractéristiques ARNs, reportez-vous à la section [Ressources définies par AWS X-Ray](#) dans le guide de l'utilisateur IAM. Pour savoir grâce à quelles actions vous pouvez spécifier l'ARN de chaque ressource, consultez [Actions définies par AWS X-Ray](#).

Clés de condition

X-Ray ne fournit aucune clé de condition spécifique à un service, mais prend en charge l'utilisation de certaines clés de condition globales. Pour voir toutes les clés de condition AWS globales, consultez la section [Clés contextuelles de condition AWS globale](#) dans le guide de l'utilisateur IAM.

Exemples

Pour consulter des exemples de politiques basées sur l'identité de X-Ray, consultez. [AWS X-Ray exemples de politiques basées sur l'identité](#)

Politiques basées sur les ressources de X-Ray

X-Ray prend en charge les politiques basées sur les ressources pour l' Service AWS intégration actuelle et future, telles que le suivi actif [Amazon SNS](#). Les politiques basées sur les ressources de X-Ray peuvent être mises à jour par d'autres AWS Management Console utilisateurs, via le AWS SDK ou la CLI. Par exemple, la console Amazon SNS tente de configurer automatiquement

une politique basée sur les ressources pour envoyer des traces à X-Ray. Le document de politique suivant fournit un exemple de configuration manuelle de la politique basée sur les ressources de X-Ray.

Exemple Exemple de politique basée sur les ressources X-Ray pour le suivi actif d'Amazon SNS

Cet exemple de document de politique précise les autorisations dont Amazon SNS a besoin pour envoyer des données de suivi à X-Ray :

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

Utilisez la CLI pour créer une politique basée sur les ressources qui autorise Amazon SNS à envoyer des données de suivi à X-Ray :

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

Pour utiliser ces exemples, remplacez *partition*, *region*, *account-id*, et *topic-name* par votre AWS partition, région, ID de compte et nom de rubrique Amazon SNS spécifiques. Pour autoriser toutes les rubriques Amazon SNS à envoyer des données de suivi à X-Ray, remplacez le nom de la rubrique par. *

Autorisation basée sur les tags X-Ray

Vous pouvez associer des tags à des groupes X-Ray ou à des règles d'échantillonnage, ou transmettre des tags dans une demande à X-Ray. Pour contrôler l'accès basé sur des étiquettes, vous devez fournir les informations d'étiquette dans l'[élément de condition](#) d'une politique utilisant les clés de condition `xray:ResourceTag/key-name`, `aws:RequestTag/key-name` ou `aws:TagKeys`. Pour plus d'informations sur le balisage des ressources X-Ray, consultez [Étiquetage des règles et des groupes d'échantillonnage aux rayons X](#).

Pour visualiser un exemple de politique basée sur l'identité permettant de limiter l'accès à une ressource en fonction des balises de cette ressource, consultez [Gestion de l'accès aux groupes X-Ray et des règles d'échantillonnage en fonction des balises](#).

Exécution de votre application en local

Votre application instrumentée envoie des données de trace au daemon X-Ray. Le daemon met en mémoire tampon les documents et les télécharge par lots vers le service X-Ray. Le daemon a besoin d'autorisations d'écriture pour télécharger des données de trace et des données de télémétrie vers le service X-Ray.

Lorsque vous [exécutez le démon localement](#), créez un rôle IAM, [assumez le rôle](#) et stockez les informations d'identification temporaires dans des variables d'environnement ou dans un fichier nommé `credentials` dans un dossier nommé `.aws` dans votre dossier utilisateur. Pour plus d'informations, reportez-vous à la section [Utilisation d'informations d'identification AWS CLI de sécurité temporaires avec le](#).

Exemple `~/aws/credentials`

```
[default]
aws_access_key_id={access key ID}
aws_secret_access_key={access key}
```

```
aws_session_token={AWS session token}
```

Si vous avez déjà configuré des informations d'identification à utiliser avec le AWS SDK ou AWS CLI, le démon peut les utiliser. Si plusieurs profils sont disponibles, le démon utilise celui par défaut.

Exécution de votre application dans AWS

Lorsque vous exécutez votre application sur AWS, utilisez un rôle pour accorder l'autorisation à l'EC2 instance Amazon ou à la fonction Lambda qui exécute le démon.

- Amazon Elastic Compute Cloud (Amazon EC2) : créez un rôle IAM et associez-le à l'EC2 instance en tant que [profil d'instance](#).
- Amazon Elastic Container Service (Amazon ECS) — Créez un rôle IAM et associez-le aux instances de conteneur en tant que rôle IAM [d'instance de conteneur](#).
- AWS Elastic Beanstalk (Elastic Beanstalk) — Elastic [Beanstalk inclut les autorisations X-Ray dans son profil d'instance par défaut](#). Vous pouvez utiliser le profil d'instance par défaut ou ajouter des autorisations d'écriture à un profil d'instance personnalisé.
- AWS Lambda (Lambda) — Ajoutez des autorisations d'écriture au rôle d'exécution de votre fonction.

Pour créer un rôle à utiliser avec X-Ray

1. Ouvrez la [console IAM](#).
2. Sélectionnez Roles (Rôles).
3. Choisissez Create New Role (Créer un nouveau rôle).
4. Sous Nom du rôle, tapez **xray-application**. Choisissez Étape suivante.
5. Pour Type de rôle, choisissez Amazon EC2.
6. Joignez la politique gérée suivante pour permettre à votre application d'accéder à Services AWS :
 - AWSXRayDaemonWriteAccess— Donne au daemon X-Ray l'autorisation de télécharger des données de trace.

Si votre application utilise le AWS SDK pour accéder à d'autres services, ajoutez des politiques qui accordent l'accès à ces services.

7. Choisissez Étape suivante.

8. Choisissez Create Role (Créer le rôle).

Autorisations utilisateur pour le chiffrement

X-Ray chiffre toutes les données de trace et, par défaut, vous pouvez le [configurer pour utiliser une clé que vous gérez](#). Si vous choisissez une clé gérée par le AWS Key Management Service client, vous devez vous assurer que la politique d'accès de la clé vous permet d'autoriser X-Ray à l'utiliser pour le chiffrement. Les autres utilisateurs de votre compte doivent également accéder à la clé pour consulter les données de suivi cryptées dans la console X-Ray.

Pour une clé gérée par le client, configurez votre clé avec une politique d'accès qui autorise les actions suivantes :

- L'utilisateur qui configure la clé dans X-Ray est autorisé à appeler `kms:CreateGrant` et `kms:DescribeKey`.
- Les utilisateurs qui peuvent accéder à des données de suivi chiffrées sont autorisés à appeler `kms:Decrypt`.

Lorsque vous ajoutez un utilisateur au groupe Utilisateurs clés dans la section de configuration des clés de la console IAM, il est autorisé à effectuer ces deux opérations. Les autorisations doivent uniquement être définies selon la politique clé, vous n'avez donc pas besoin d' AWS KMS autorisations pour vos utilisateurs, groupes ou rôles. Pour plus d'informations, consultez la section [Utilisation des politiques clés dans le guide du AWS KMS développeur](#).

Pour le chiffrement par défaut, ou si vous choisissez la AWS clé CMK gérée (`aws/xray`), l'autorisation dépend de la personne ayant accès à X-Ray APIs. Toute personne ayant accès à [PutEncryptionConfig](#), incluse dans `AWSXrayFullAccess`, peut modifier la configuration de chiffrement. Pour empêcher un utilisateur de changer la clé de chiffrement, ne leur donnez pas l'autorisation d'utiliser [PutEncryptionConfig](#).

AWS X-Ray exemples de politiques basées sur l'identité

Par défaut, les utilisateurs et les rôles ne sont pas autorisés à créer ou à modifier les ressources X-Ray. Ils ne peuvent pas non plus effectuer de tâches à l'aide de l' AWS API AWS Management Console AWS CLI, ou. Un administrateur doit créer des politiques IAM autorisant les utilisateurs et les rôles à exécuter des opérations d'API spécifiques sur les ressources spécifiées dont ils ont besoin. Il doit ensuite attacher ces stratégies aux utilisateurs ou aux groupes ayant besoin de ces autorisations.

Pour savoir comment créer une politique IAM basée sur l'identité à l'aide de ces exemples de documents de politique JSON, consultez [Création de politiques dans l'onglet JSON](#) dans le Guide de l'utilisateur IAM.

Rubriques

- [Bonnes pratiques en matière de politiques](#)
- [Utilisation de la console X-Ray](#)
- [Autorisation accordée aux utilisateurs pour afficher leurs propres autorisations](#)
- [Gestion de l'accès aux groupes X-Ray et des règles d'échantillonnage en fonction des balises](#)
- [Politiques gérées par IAM pour X-Ray](#)
- [X-Ray met à jour les politiques AWS gérées](#)
- [Spécification d'une ressource au sein d'une politique IAM](#)

Bonnes pratiques en matière de politiques

Les politiques basées sur l'identité déterminent si quelqu'un peut créer, accéder ou supprimer des ressources X-Ray dans votre compte. Ces actions peuvent entraîner des frais pour votre Compte AWS. Lorsque vous créez ou modifiez des politiques basées sur l'identité, suivez ces instructions et recommandations :

- Commencez AWS par les politiques gérées et passez aux autorisations du moindre privilège : pour commencer à accorder des autorisations à vos utilisateurs et à vos charges de travail, utilisez les politiques AWS gérées qui accordent des autorisations pour de nombreux cas d'utilisation courants. Ils sont disponibles dans votre Compte AWS. Nous vous recommandons de réduire davantage les autorisations en définissant des politiques gérées par les AWS clients spécifiques à vos cas d'utilisation. Pour plus d'informations, consultez [politiques gérées par AWS](#) ou [politiques gérées par AWS pour les activités professionnelles](#) dans le Guide de l'utilisateur IAM.
- Accordez les autorisations de moindre privilège : lorsque vous définissez des autorisations avec des politiques IAM, accordez uniquement les autorisations nécessaires à l'exécution d'une seule tâche. Pour ce faire, vous définissez les actions qui peuvent être entreprises sur des ressources spécifiques dans des conditions spécifiques, également appelées autorisations de moindre privilège. Pour plus d'informations sur l'utilisation d'IAM pour appliquer des autorisations, consultez [politiques et autorisations dans IAM](#) dans le Guide de l'utilisateur IAM.
- Utilisez des conditions dans les politiques IAM pour restreindre davantage l'accès : vous pouvez ajouter une condition à vos politiques afin de limiter l'accès aux actions et aux ressources. Par

exemple, vous pouvez écrire une condition de politique pour spécifier que toutes les demandes doivent être envoyées via SSL. Vous pouvez également utiliser des conditions pour accorder l'accès aux actions de service si elles sont utilisées par le biais d'un service spécifique Service AWS, tel que AWS CloudFormation. Pour plus d'informations, consultez [Conditions pour éléments de politique JSON IAM](#) dans le Guide de l'utilisateur IAM.

- Utilisez l'Analyseur d'accès IAM pour valider vos politiques IAM afin de garantir des autorisations sécurisées et fonctionnelles : l'Analyseur d'accès IAM valide les politiques nouvelles et existantes de manière à ce que les politiques IAM respectent le langage de politique IAM (JSON) et les bonnes pratiques IAM. IAM Access Analyzer fournit plus de 100 vérifications de politiques et des recommandations exploitables pour vous aider à créer des politiques sécurisées et fonctionnelles. Pour plus d'informations, consultez [Validation de politiques avec IAM Access Analyzer](#) dans le Guide de l'utilisateur IAM.
- Exiger l'authentification multifactorielle (MFA) : si vous avez un scénario qui nécessite des utilisateurs IAM ou un utilisateur root, activez l'authentification MFA pour une sécurité accrue. Compte AWS Pour exiger la MFA lorsque des opérations d'API sont appelées, ajoutez des conditions MFA à vos politiques. Pour plus d'informations, consultez [Sécurisation de l'accès aux API avec MFA](#) dans le Guide de l'utilisateur IAM.

Pour plus d'informations sur les bonnes pratiques dans IAM, consultez [Bonnes pratiques de sécurité dans IAM](#) dans le Guide de l'utilisateur IAM.

Utilisation de la console X-Ray

Pour accéder à la AWS X-Ray console, vous devez disposer d'un ensemble minimal d'autorisations. Ces autorisations doivent vous permettre de répertorier et de consulter les détails des ressources X-Ray de votre Compte AWS. Si vous créez une politique basée sur l'identité qui est plus restrictive que l'ensemble minimum d'autorisations requis, la console ne fonctionnera pas comme prévu pour les entités (utilisateurs ou rôles) tributaires de cette politique.

Pour vous assurer que ces entités peuvent toujours utiliser la console X-Ray, associez la politique `AWSXRayReadOnlyAccess` AWS gérée aux entités. Cette politique est décrite plus en détail dans [Politiques gérées par IAM pour X-Ray](#). Pour plus d'informations, consultez [Ajout d'autorisations à un utilisateur](#) dans le Guide de l'utilisateur IAM.

Il n'est pas nécessaire d'accorder des autorisations de console minimales aux utilisateurs qui appellent uniquement l'API AWS CLI ou l' AWS API. Autorisez plutôt l'accès à uniquement aux actions qui correspondent à l'opération d'API que vous tentez d'effectuer.

Autorisation accordée aux utilisateurs pour afficher leurs propres autorisations

Cet exemple montre comment créer une politique qui permet aux utilisateurs IAM d'afficher les politiques en ligne et gérées attachées à leur identité d'utilisateur. Cette politique inclut les autorisations permettant d'effectuer cette action sur la console ou par programmation à l'aide de l'API AWS CLI or AWS .

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ViewOwnUserInfo",
      "Effect": "Allow",
      "Action": [
        "iam:GetUserPolicy",
        "iam:ListGroupsForUser",
        "iam:ListAttachedUserPolicies",
        "iam:ListUserPolicies",
        "iam:GetUser"
      ],
      "Resource": ["arn:aws:iam::*:user/${aws:username}"]
    },
    {
      "Sid": "NavigateInConsole",
      "Effect": "Allow",
      "Action": [
        "iam:GetGroupPolicy",
        "iam:GetPolicyVersion",
        "iam:GetPolicy",
        "iam:ListAttachedGroupPolicies",
        "iam:ListGroupPolicies",
        "iam:ListPolicyVersions",
        "iam:ListPolicies",
        "iam:ListUsers"
      ],
      "Resource": "*"
    }
  ]
}
```

Gestion de l'accès aux groupes X-Ray et des règles d'échantillonnage en fonction des balises

Vous pouvez utiliser les conditions de votre politique basée sur l'identité pour contrôler l'accès aux groupes X-Ray et les règles d'échantillonnage basées sur des balises. L'exemple de politique suivant peut être utilisé pour refuser à un rôle d'utilisateur l'autorisation de créer, de supprimer ou de mettre à jour des groupes avec les balises `stage:prod` ou `stage:preprod`. Pour plus d'informations sur le balisage des règles et des groupes d'échantillonnage X-Ray, consultez [Étiquetage des règles et des groupes d'échantillonnage aux rayons X](#).

Pour refuser à un utilisateur l'accès à la création, à la mise à jour ou à la suppression d'un groupe doté d'une étiquette `stage:prod` ou `stage:preprod` pour attribuer à l'utilisateur un rôle avec une politique similaire à la suivante.

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

```

Pour refuser la création d'une règle d'échantillonnage, utilisez cette option `aws:RequestTag` pour indiquer les balises qui ne peuvent pas être transmises dans le cadre d'une demande de création. Pour refuser la mise à jour ou la suppression d'une règle d'échantillonnage, utilisez cette `aws:ResourceTag` option pour refuser les actions en fonction des balises associées à ces ressources.

```

{
  "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"
          ]
        }
      }
    }
  ]
}

```

Vous pouvez associer ces politiques (ou les combiner en une seule politique, puis associer la politique) aux utilisateurs de votre compte. Pour que l'utilisateur puisse modifier un groupe ou une règle d'échantillonnage, le groupe ou la règle d'échantillonnage ne doit pas être balisé `stage=preprod` ou `stage=prod`. La clé de condition d'étiquette `Stage` correspond à la fois à `Stage` et à `stage`, car les noms de clé de condition ne sont pas sensibles à la casse. Pour plus d'informations sur le bloc de conditions, voir [IAM JSON Policy Elements : Condition](#) dans le guide de l'utilisateur IAM.

Un utilisateur dont le rôle est associé à la politique suivante ne peut pas ajouter la balise `role:admin` aux ressources et ne peut pas supprimer de balises d'une ressource qui lui est `role:admin` associée.

```

{
  "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"
            }
        }
    }
]
}

```

Politiques gérées par IAM pour X-Ray

Pour faciliter l'octroi des autorisations, IAM prend en charge les politiques gérées pour chaque service. Un service peut mettre à jour ces politiques gérées avec de nouvelles autorisations lorsqu'il en publie de nouvelles APIs. AWS X-Ray fournit des politiques gérées pour les cas d'utilisation en lecture seule, en écriture seule et par les administrateurs.

- **AWSXrayReadOnlyAccess**— Autorisations de lecture pour utiliser la console X-Ray, ou le AWS SDK AWS CLI, afin d'obtenir des données de suivi, des cartes de traçage, des informations et la configuration de X-Ray à partir de l'API X-Ray. Inclut le gestionnaire d'accès à l'observabilité (OAM) `oam:ListSinks` et `oam:ListAttachedSinks` des autorisations permettant à la console de consulter les traces partagées depuis les comptes sources dans le cadre de l'observabilité [CloudWatch entre](#) comptes. Les actions `BatchGetTraceSummaryById` et

GetDistinctTraceGraphs API ne sont pas destinées à être appelées par votre code et ne sont pas incluses dans le AWS CLI et AWS SDKs.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "xray:GetSamplingRules",
        "xray:GetSamplingTargets",
        "xray:GetSamplingStatisticSummaries",
        "xray:BatchGetTraces",
        "xray:BatchGetTraceSummaryById",
        "xray:GetDistinctTraceGraphs",
        "xray:GetServiceGraph",
        "xray:GetTraceGraph",
        "xray:GetTraceSummaries",
        "xray:GetGroups",
        "xray:GetGroup",
        "xray:ListTagsForResource",
        "xray:ListResourcePolicies",
        "xray:GetTimeSeriesServiceStatistics",
        "xray:GetInsightSummaries",
        "xray:GetInsight",
        "xray:GetInsightEvents",
        "xray:GetInsightImpactGraph",
        "oam:ListSinks"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "oam:ListAttachedLinks"
      ],
      "Resource": "arn:aws:oam:*:*:sink/*"
    }
  ]
}
```

- **AWSXRayDaemonWriteAccess**— Écrivez des autorisations pour utiliser le daemon X-Ray, ou AWS SDK AWS CLI, pour télécharger des documents segmentés et des données de télémétrie vers l'API X-Ray. Inclut les autorisations de lecture pour obtenir les [règles d'échantillonnage](#) et rapporter les résultats d'échantillonnage.

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

- **AWSXrayCrossAccountSharingConfiguration**— Accorde les autorisations nécessaires pour créer, gérer et consulter les liens d'Observability Access Manager pour partager les ressources X-Ray entre les comptes. Utilisé pour permettre l'[observabilité entre CloudWatch comptes](#) source et comptes de surveillance.

```
{
  "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**— Autorisation d'utiliser tous les X-Ray APIs, y compris les autorisations de lecture, les autorisations d'écriture et l'autorisation de configurer les paramètres des clés de chiffrement et les règles d'échantillonnage. Inclut le gestionnaire d'accès à l'observabilité (OAM) `oam:ListSinks` et `oam:ListAttachedSinks` des autorisations permettant à la console de consulter les traces partagées depuis les comptes sources dans le cadre de l'observabilité [CloudWatch entre](#) comptes.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "xray:*",
                "oam:ListSinks"
            ],
            "Resource": [
                "*"
            ]
        },
        {

```

```

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

```

Pour ajouter une stratégie gérée à un utilisateur, groupe ou rôle IAM

1. Ouvrez la [console IAM](#).
2. Ouvrez le rôle associé à votre profil d'instance, un utilisateur IAM ou un groupe IAM.
3. Sous Autorisations, attachez la stratégie gérée.

X-Ray met à jour les politiques AWS gérées

Consultez les détails des mises à jour apportées aux politiques AWS gérées pour X-Ray depuis que ce service a commencé à suivre ces modifications. Pour recevoir des alertes automatiques concernant les modifications apportées à cette page, abonnez-vous au flux RSS sur la page d'[historique du document](#) X-Ray.

Modification	Description	Date
Politiques gérées par IAM pour X-Ray — Ajout de AWSXrayFullAccess politiques nouvelles AWSXrayCrossAccountSharingConfiguration AWSXrayReadOnlyAccess et mises à jour.	X-Ray a ajouté des autorisations oam:ListSinks Observability Access Manager (OAM) oam:ListAttachedSinks à ces politiques pour permettre à la console de visualiser les traces partagées depuis les comptes sources dans le cadre de l'observabilité CloudWatch entre comptes.	27 novembre 2022

Modification	Description	Date
Politiques gérées par IAM pour X-Ray — Mise à jour de la AWSXrayReadOnlyAccess politique	X-Ray a ajouté une action d'API, ListResourcePolicies .	15 novembre 2022
Utilisation de la console X-Ray — Mise à jour de la AWSXrayReadOnlyAccess politique	X-Ray a ajouté deux nouvelles actions d'API, BatchGetTraceSummaryById et GetDistinctTraceGraphs . Ces actions ne sont pas destinées à être appelées par votre code. Par conséquent, ces actions d'API ne sont pas incluses dans le AWS CLI et AWS SDKs.	11 novembre 2022

Spécification d'une ressource au sein d'une politique IAM

Vous pouvez contrôler l'accès aux ressources à l'aide d'une politique IAM. Pour les actions qui prennent en charge les autorisations au niveau des ressources, vous utilisez un Amazon Resource Name (ARN) pour identifier la ressource à laquelle la stratégie s'applique.

Toutes les actions X-Ray peuvent être utilisées dans une politique IAM pour accorder ou refuser aux utilisateurs l'autorisation d'utiliser cette action. Cependant, les [actions X-Ray](#) ne prennent pas toutes en charge les autorisations au niveau des ressources, qui vous permettent de spécifier les ressources sur lesquelles une action peut être effectuée.

Pour les actions qui ne prennent pas en charge les autorisations de niveau ressource, vous devez utiliser « * » comme ressource.

Les actions X-Ray suivantes prennent en charge les autorisations au niveau des ressources :

- CreateGroup
- GetGroup

- UpdateGroup
- DeleteGroup
- CreateSamplingRule
- UpdateSamplingRule
- DeleteSamplingRule

Vous trouverez ci-dessous un exemple de stratégie d'autorisations basées sur l'identité pour une action `CreateGroup`. Cet exemple montre comment se servir d'un ARN lié à un nom de groupe `local-users`, en utilisant l'ID unique en tant que caractère générique. L'ID unique est généré lorsque le groupe est créé. Il ne peut donc pas être prédit à l'avance dans la stratégie. Lorsque vous utilisez `GetGroup`, `UpdateGroup` ou `DeleteGroup`, vous pouvez définir l'ID unique en tant que caractère générique ou ARN exact, comprenant l'ID.

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

Vous trouverez ci-dessous un exemple de stratégie d'autorisations basées sur l'identité pour une action `CreateSamplingRule`.

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

```

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

Note

L'ARN d'une règle d'échantillonnage est défini par le nom de cette dernière. Contrairement aux règles de groupe ARNs, les règles d'échantillonnage n'ont pas d'identifiant généré de manière unique.

Résolution des problèmes AWS X-Ray d'identité et d'accès

Utilisez les informations suivantes pour vous aider à diagnostiquer et à résoudre les problèmes courants que vous pouvez rencontrer lorsque vous travaillez avec X-Ray et IAM.

Rubriques

- [Je ne suis pas autorisé à effectuer une action dans X-Ray](#)
- [Je ne suis pas autorisé à effectuer iam : PassRole](#)
- [Je suis administrateur et je souhaite autoriser d'autres personnes à accéder à X-Ray](#)
- [Je veux permettre à des personnes extérieures Compte AWS à moi d'accéder à mes ressources X-Ray](#)

Je ne suis pas autorisé à effectuer une action dans X-Ray

S'il vous AWS Management Console indique que vous n'êtes pas autorisé à effectuer une action, vous devez contacter votre administrateur pour obtenir de l'aide. Votre administrateur est la personne qui vous a fourni vos informations de connexion.

L'exemple d'erreur suivant se produit lorsque l'utilisateur `mateojackson` essaie d'utiliser la console pour afficher les détails d'une règle d'échantillonnage mais ne dispose pas des `xray:GetSamplingRules` autorisations nécessaires.

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

Dans ce cas, Mateo demande à son administrateur de mettre à jour ses stratégies pour l'autoriser à accéder à la ressource de la règle d'échantillonnage à l'aide de l'action `xray:GetSamplingRules`.

Je ne suis pas autorisé à effectuer `iam:PassRole`

Si vous recevez un message d'erreur indiquant que vous n'êtes pas autorisé à effectuer l'`iam:PassRole` action, vos politiques doivent être mises à jour pour vous permettre de transmettre un rôle à X-Ray.

Certains services AWS permettent de transmettre un rôle existant à ce service au lieu de créer un nouveau rôle de service ou un rôle lié à un service. Pour ce faire, un utilisateur doit disposer des autorisations nécessaires pour transmettre le rôle au service.

L'exemple d'erreur suivant se produit lorsqu'un utilisateur IAM nommé `marymajor` essaie d'utiliser la console pour effectuer une action dans X-Ray. Toutefois, l'action nécessite que le service ait des autorisations accordées par un rôle de service. Mary ne dispose pas des autorisations nécessaires pour transférer le rôle au service.

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

Dans ce cas, les politiques de Mary doivent être mises à jour pour lui permettre d'exécuter l'action `iam:PassRole`.

Si vous avez besoin d'aide, contactez votre AWS administrateur. Votre administrateur vous a fourni vos informations d'identification de connexion.

Je suis administrateur et je souhaite autoriser d'autres personnes à accéder à X-Ray

Pour autoriser d'autres personnes à accéder à X-Ray, vous devez autoriser les personnes ou les applications qui ont besoin d'y accéder. Si vous utilisez AWS IAM Identity Center pour gérer des personnes et des applications, vous attribuez des ensembles d'autorisations aux utilisateurs ou aux groupes afin de définir leur niveau d'accès. Les ensembles d'autorisations créent et attribuent automatiquement des politiques IAM aux rôles IAM associés à la personne ou à l'application. Pour

plus d'informations, consultez la section [Ensembles d'autorisations](#) dans le guide de AWS IAM Identity Center l'utilisateur.

Si vous n'utilisez pas IAM Identity Center, vous devez créer des entités IAM (utilisateurs ou rôles) pour les personnes ou les applications qui ont besoin d'un accès. Vous devez ensuite associer une politique à l'entité qui lui accorde les autorisations appropriées dans X-Ray. Une fois les autorisations accordées, fournissez les informations d'identification à l'utilisateur ou au développeur de l'application. Ils utiliseront ces informations d'identification pour y accéder AWS. Pour en savoir plus sur la création d'utilisateurs, de groupes, de politiques et d'autorisations [IAM, consultez la section Identités, politiques et autorisations IAM dans le guide de l'utilisateur IAM](#).

Je veux permettre à des personnes extérieures Compte AWS à moi d'accéder à mes ressources X-Ray

Vous pouvez créer un rôle que les utilisateurs provenant d'autres comptes ou les personnes extérieures à votre organisation pourront utiliser pour accéder à vos ressources. Vous pouvez spécifier qui est autorisé à assumer le rôle. Pour les services qui prennent en charge les politiques basées sur les ressources ou les listes de contrôle d'accès (ACLs), vous pouvez utiliser ces politiques pour autoriser les utilisateurs à accéder à vos ressources.

Pour plus d'informations, consultez les éléments suivants :

- Pour savoir si X-Ray prend en charge ces fonctionnalités, consultez [Comment AWS X-Ray fonctionne avec IAM](#).
- Pour savoir comment fournir l'accès à vos ressources sur celles Comptes AWS que vous possédez, consultez la section [Fournir l'accès à un utilisateur IAM dans un autre utilisateur Compte AWS que vous possédez](#) dans le Guide de l'utilisateur IAM.
- Pour savoir comment fournir l'accès à vos ressources à des tiers Comptes AWS, consultez la section [Fournir un accès à des ressources Comptes AWS détenues par des tiers](#) dans le guide de l'utilisateur IAM.
- Pour savoir comment fournir un accès par le biais de la fédération d'identité, consultez [Fournir un accès à des utilisateurs authentifiés en externe \(fédération d'identité\)](#) dans le Guide de l'utilisateur IAM.
- Pour en savoir plus sur la différence entre l'utilisation des rôles et des politiques basées sur les ressources pour l'accès intercompte, consultez [Accès intercompte aux ressources dans IAM](#) dans le Guide de l'utilisateur IAM.

Connexion et surveillance AWS X-Ray

La surveillance est essentielle pour assurer la fiabilité, la disponibilité et les performances de vos solutions AWS . Vous devez collecter des données de surveillance provenant de toutes les parties de votre AWS solution afin de pouvoir corriger plus facilement une défaillance multipoint, le cas échéant. AWS fournit plusieurs outils pour surveiller vos ressources X-Ray et répondre aux incidents potentiels :

AWS CloudTrail Journaux

AWS X-Ray s'intègre AWS CloudTrail pour enregistrer les actions d'API effectuées par un utilisateur, un rôle ou un AWS service dans X-Ray. Vous pouvez l'utiliser CloudTrail pour surveiller les demandes d'API X-Ray en temps réel et stocker les journaux dans Amazon S3, Amazon CloudWatch Logs et Amazon CloudWatch Events. Pour de plus amples informations, veuillez consulter [Enregistrement des appels d'API X-Ray avec AWS CloudTrail](#).

AWS Config Suivi

AWS X-Ray s'intègre AWS Config pour enregistrer les modifications de configuration apportées à vos ressources de chiffrement X-Ray. Vous pouvez l'utiliser AWS Config pour inventorier les ressources de chiffrement de X-Ray, vérifier l'historique de configuration de X-Ray et envoyer des notifications en fonction des modifications apportées aux ressources. Pour de plus amples informations, veuillez consulter [Suivi des modifications de configuration du chiffrement X-Ray avec AWS Config](#).

CloudWatch Surveillance d'Amazon

Vous pouvez utiliser le SDK X-Ray pour Java pour publier des métriques CloudWatch Amazon non échantillonnées à partir des segments X-Ray que vous avez collectés. Ces métriques sont dérivées de l'heure de début et de fin du segment, ainsi que des indicateurs d'erreur, de défaut et d'état de limitation. Utilisez ces métriques de suivi pour exposer les nouvelles tentatives et les problèmes de dépendance dans les sous-segments. Pour de plus amples informations, veuillez consulter [AWS X-Ray métriques pour le SDK X-Ray pour Java](#).

Validation de conformité pour AWS X-Ray

Pour savoir si un [programme Services AWS de conformité Service AWS s'inscrit dans le champ d'application de programmes de conformité](#) spécifiques, consultez Services AWS la section de

conformité et sélectionnez le programme de conformité qui vous intéresse. Pour des informations générales, voir Programmes de [AWS conformité Programmes AWS](#) de .

Vous pouvez télécharger des rapports d'audit tiers à l'aide de AWS Artifact. Pour plus d'informations, voir [Téléchargement de rapports dans AWS Artifact](#) .

Votre responsabilité en matière de conformité lors de l'utilisation Services AWS est déterminée par la sensibilité de vos données, les objectifs de conformité de votre entreprise et les lois et réglementations applicables. AWS fournit les ressources suivantes pour faciliter la mise en conformité :

- [Conformité et gouvernance de la sécurité](#) : ces guides de mise en œuvre de solutions traitent des considérations architecturales et fournissent les étapes à suivre afin de déployer des fonctionnalités de sécurité et de conformité.
- [Référence des services éligibles HIPAA](#) : liste les services éligibles HIPAA. Tous ne Services AWS sont pas éligibles à la loi HIPAA.
- AWS Ressources de <https://aws.amazon.com/compliance/resources/> de conformité — Cette collection de classeurs et de guides peut s'appliquer à votre secteur d'activité et à votre région.
- [AWS Guides de conformité destinés aux clients](#) — Comprenez le modèle de responsabilité partagée sous l'angle de la conformité. Les guides résument les meilleures pratiques en matière de sécurisation Services AWS et décrivent les directives relatives aux contrôles de sécurité dans plusieurs cadres (notamment le National Institute of Standards and Technology (NIST), le Payment Card Industry Security Standards Council (PCI) et l'Organisation internationale de normalisation (ISO)).
- [Évaluation des ressources à l'aide des règles](#) du guide du AWS Config développeur : le AWS Config service évalue dans quelle mesure les configurations de vos ressources sont conformes aux pratiques internes, aux directives du secteur et aux réglementations.
- [AWS Security Hub](#)— Cela Service AWS fournit une vue complète de votre état de sécurité interne AWS. Security Hub utilise des contrôles de sécurité pour évaluer vos ressources AWS et vérifier votre conformité par rapport aux normes et aux bonnes pratiques du secteur de la sécurité. Pour obtenir la liste des services et des contrôles pris en charge, consultez [Référence des contrôles Security Hub](#).
- [Amazon GuardDuty](#) — Cela Service AWS détecte les menaces potentielles qui pèsent sur vos charges de travail Comptes AWS, vos conteneurs et vos données en surveillant votre environnement pour détecter toute activité suspecte et malveillante. GuardDuty peut vous aider à répondre à diverses exigences de conformité, telles que la norme PCI DSS, en répondant aux exigences de détection des intrusions imposées par certains cadres de conformité.

- [AWS Audit Manager](#)— Cela vous permet Service AWS d'auditer en permanence votre AWS utilisation afin de simplifier la gestion des risques et la conformité aux réglementations et aux normes du secteur.

Résilience dans AWS X-Ray

L'infrastructure AWS mondiale est construite autour Régions AWS de zones de disponibilité. Régions AWS fournissent plusieurs zones de disponibilité physiquement séparées et isolées, connectées par un réseau à faible latence, à haut débit et hautement redondant. Avec les zones de disponibilité, vous pouvez concevoir et exploiter des applications et des bases de données qui basculent automatiquement d'une zone de disponibilité à l'autre sans interruption. Les zones de disponibilité sont plus hautement disponibles, tolérantes aux pannes et évolutives que les infrastructures traditionnelles à un ou plusieurs centres de données.

Pour plus d'informations sur les zones de disponibilité Régions AWS et les zones de disponibilité, consultez la section [Infrastructure AWS globale](#).

Sécurité de l'infrastructure dans AWS X-Ray

En tant que service géré, AWS X-Ray il est protégé par la sécurité du réseau AWS mondial. Pour plus d'informations sur les services AWS de sécurité et sur la manière dont AWS l'infrastructure est protégée, consultez la section [Sécurité du AWS cloud](#). Pour concevoir votre AWS environnement en utilisant les meilleures pratiques en matière de sécurité de l'infrastructure, consultez la section [Protection de l'infrastructure](#) dans le cadre AWS bien architecturé du pilier de sécurité.

Vous utilisez des appels d'API AWS publiés pour accéder à X-Ray via le réseau. Les clients doivent prendre en charge les éléments suivants :

- Protocole TLS (Transport Layer Security). Nous exigeons TLS 1.2 et recommandons TLS 1.3.
- Ses suites de chiffrement PFS (Perfect Forward Secrecy) comme DHE (Ephemeral Diffie-Hellman) ou ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). La plupart des systèmes modernes tels que Java 7 et les versions ultérieures prennent en charge ces modes.

En outre, les demandes doivent être signées à l'aide d'un ID de clé d'accès et d'une clé d'accès secrète associée à un principal IAM. Vous pouvez également utiliser [AWS Security Token Service](#) (AWS STS) pour générer des informations d'identification de sécurité temporaires et signer les demandes.

Utilisation AWS X-Ray avec des points de terminaison VPC

Si vous utilisez Amazon Virtual Private Cloud (Amazon VPC) pour héberger vos AWS ressources, vous pouvez établir une connexion privée entre votre VPC et X-Ray. Cela permet aux ressources de votre Amazon VPC de communiquer avec le service X-Ray sans passer par l'Internet public.

Amazon VPC est un système Service AWS que vous pouvez utiliser pour lancer AWS des ressources dans un réseau virtuel que vous définissez. Avec un VPC, vous contrôlez des paramètres réseau, tels que la plage d'adresses IP, les sous-réseaux, les tables de routage et les passerelles réseau. Pour connecter votre VPC à X-Ray, vous devez définir un point de terminaison [VPC d'interface](#). Le point de terminaison fournit une connectivité fiable et évolutive à X-Ray sans nécessiter de passerelle Internet, d'instance de traduction d'adresses réseau (NAT) ou de connexion VPN. Pour de plus amples informations, veuillez consulter [Qu'est-ce qu'Amazon VPC ?](#) dans le Guide de l'utilisateur Amazon VPC.

Les points de terminaison VPC d'interface sont alimentés par AWS PrivateLink une AWS technologie qui permet une communication privée entre eux Services AWS en utilisant une interface Elastic Network avec des adresses IP privées. Pour plus d'informations, consultez le billet de Services AWS blog [New — AWS PrivateLink for](#) et [Getting Started](#) dans le guide de l'utilisateur Amazon VPC.

Pour vous assurer que vous pouvez créer un point de terminaison VPC pour X-Ray dans celui de votre choix Région AWS, consultez. [Régions prises en charge](#)

Création d'un point de terminaison VPC pour X-Ray

Pour commencer à utiliser X-Ray avec votre VPC, créez un point de terminaison VPC d'interface pour X-Ray.

1. Ouvrez la console Amazon VPC à l'adresse <https://console.aws.amazon.com/vpc/>.
2. Accédez à Endpoints dans le volet de navigation et choisissez Create Endpoint.
3. Recherchez et sélectionnez le nom du AWS X-Ray service :com.amazonaws.**region**.xray.

Service category ☒ AWS services
☐ Find service by name
☐ Your AWS Marketplace services

Service Name com.amazonaws.us-west-2.xray ⓘ

	Service Name	Owner	Type
☐	com.amazonaws.us-west-2.transfer.server	amazon	Interface
☐	com.amazonaws.us-west-2.workspaces	amazon	Interface
☒	com.amazonaws.us-west-2.xray	amazon	Interface

4. Sélectionnez le VPC de votre choix, puis sélectionnez un sous-réseau dans votre VPC pour utiliser le point de terminaison de l'interface. Une interface réseau de point de terminaison est créée dans le sous-réseau sélectionné. Vous pouvez spécifier plusieurs sous-réseaux dans différentes zones de disponibilité (telles que prises en charge par le service) afin de vous assurer que le point de terminaison de votre interface résiste aux défaillances des zones de disponibilité. Dans ce cas, une interface réseau est créée dans chaque sous-réseau que vous spécifiez.

VPC* vpc-4f6e3a37 ⓘ ⓘ

Subnets subnet-40d87938 ⓘ

	Availability Zone	Subnet ID	
☒	us-west-2a (usw2-az1)	subnet-40d87938	▼
☐	us-west-2b (usw2-az2)	subnet-ff4281b5	▼
☐	us-west-2c (usw2-az3)	subnet-d14bfb8c	▼
☐	us-west-2d (usw2-az4)	subnet-1faf8734	▼

5. (Facultatif) Le DNS privé est activé par défaut pour le point de terminaison, afin que vous puissiez envoyer des demandes à X-Ray en utilisant son nom d'hôte DNS par défaut. Vous pouvez choisir de le désactiver.
6. Spécifiez les groupes de sécurité à associer à l'interface réseau du point de terminaison.

Security group

sg-d4f14ff4

[Create a new security group](#)

Select security groups ▲

Filter by tags and attributes or search by keyword

1 to 5 of 5

☐	Group ID	Group Name	VPC ID		Description	Owner ID
☐	sg-0683c...	ssh-http	vpc-4f6e3a37	EC2-VPC	launch-wizar...	979300271395
☐	sg-0774...	awseb-e-7xv5...	vpc-4f6e3a37	EC2-VPC	SecurityGrou...	979300271395
☐	sg-0a46...	launch-wizard-1	vpc-4f6e3a37	EC2-VPC	launch-wizar...	979300271395
☐	sg-0d62...	awseb-e-7xv5...	vpc-4f6e3a37	EC2-VPC	Elastic Beans...	979300271395
☒	sg-d4f14...	default	vpc-4f6e3a37	EC2-VPC	default VPC s...	979300271395

Close

- (Facultatif) Spécifiez une politique personnalisée pour contrôler les autorisations d'accès au service X-Ray. Par défaut, l'accès complet est autorisé.

Contrôle de l'accès à votre point de terminaison X-Ray VPC

Une stratégie de point de terminaison d'un VPC est une stratégie de ressource IAM que vous attachez à un point de terminaison lorsque vous le créez ou le modifiez. Si vous n'attachez pas de stratégie quand vous créez un point de terminaison, Amazon VPC attache une stratégie par défaut pour vous qui autorise un accès total au service. Une politique de point de terminaison n'annule pas et ne remplace pas les politiques utilisateur IAM ou les politiques propres au service. Il s'agit d'une politique distincte qui contrôle l'accès depuis le point de terminaison jusqu'au service spécifié. Les politiques de point de terminaison doivent être écrites au format JSON. Pour en savoir plus, consultez [Contrôle de l'accès aux services avec des points de terminaison d'un VPC](#) dans le guide de l'utilisateur Amazon VPC.

La politique des points de terminaison VPC vous permet de contrôler les autorisations relatives à diverses actions X-Ray. Par exemple, vous pouvez créer une politique pour autoriser uniquement PutTraceSegment et refuser toutes les autres actions. Cela empêche les charges de travail et les services du VPC d'envoyer uniquement des données de suivi à X-Ray et de refuser toute autre action telle que la récupération de données, la modification de la configuration de chiffrement ou la création/mise à jour de groupes.

Voici un exemple de politique de point de terminaison pour X-Ray. Cette politique permet aux utilisateurs qui se connectent à X-Ray via le VPC d'envoyer des données de segment à X-Ray et les empêche également d'effectuer d'autres actions de X-Ray.

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

Pour modifier la politique de point de terminaison VPC pour X-Ray

1. Ouvrez la console Amazon VPC à l'adresse <https://console.aws.amazon.com/vpc/>.
2. Dans le volet de navigation, choisissez Points de terminaison.
3. Si vous n'avez pas encore créé le point de terminaison pour X-Ray, suivez les étapes décrites dans [Création d'un point de terminaison VPC pour X-Ray](#).
4. Sélectionnez com.amazonaws. **region** point de terminaison .xray, puis choisissez l'onglet Policy.
5. Choisissez Edit Policy (Modifier la politique), puis apportez vos modifications.

Régions prises en charge

X-Ray prend actuellement en charge les points de terminaison VPC dans les domaines suivants :

Régions AWS

- USA Est (Ohio)
- USA Est (Virginie du Nord)
- USA Ouest (Californie du Nord)
- USA Ouest (Oregon)
- Afrique (Le Cap)
- Asie-Pacifique (Hong Kong)

- Asia Pacific (Mumbai)
- Asie-Pacifique (Osaka)
- Asia Pacific (Seoul)
- Asie-Pacifique (Singapour)
- Asie-Pacifique (Sydney)
- Asie-Pacifique (Tokyo)
- Canada (Centre)
- Europe (Francfort)
- Europe (Irlande)
- Europe (Londres)
- Europe (Milan)
- Europe (Paris)
- Europe (Stockholm)
- Moyen-Orient (Bahreïn)
- Amérique du Sud (São Paulo)
- AWS GovCloud (USA Est)
- AWS GovCloud (US-Ouest)

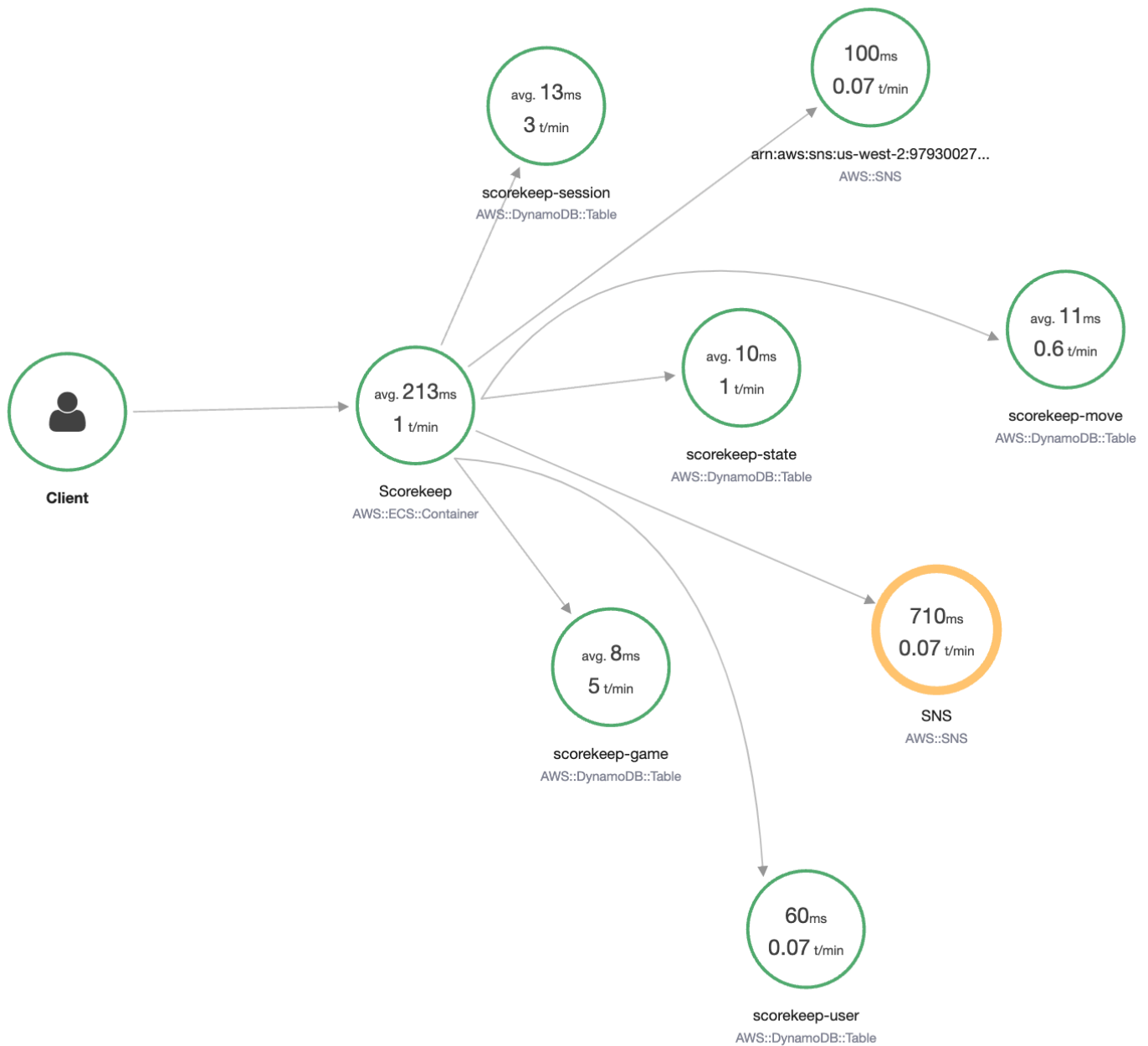
AWS X-Ray exemple d'application

L'application AWS X-Ray [eb-java-scorekeep](#) Sample, disponible sur GitHub, montre l'utilisation du SDK AWS X-Ray pour instrumenter les appels HTTP entrants, les clients du SDK DynamoDB et les clients HTTP. L'exemple d'application permet AWS CloudFormation de créer des tables DynamoDB, de compiler du code Java sur instance et d'exécuter le daemon X-Ray sans aucune configuration supplémentaire.

Consultez le [didacticiel Scorekeep](#) pour commencer à installer et à utiliser un exemple d'application instrumenté, en utilisant le ou le AWS Management Console . AWS CLI



L'exemple inclut une application Web frontale, l'API qu'elle appelle et les tables DynamoDB qu'elle utilise pour stocker des données. L'instrumentation de base avec des [filtres](#), [des plugins](#) et des [clients AWS SDK instrumentés](#) est présentée dans la branche du `xray-gettingstarted` projet. Il s'agit de la branche que vous déployez dans le [didacticiel de mise en route](#). Dans la mesure où cette branche inclut uniquement les éléments de base, vous pouvez la comparer à la branche `master` pour comprendre rapidement les éléments de base.



L'exemple d'application montre l'instrumentation de base dans ces fichiers :

- Filtre de requête HTTP — [WebConfig.java](#)
- AWS Instrumentation client du SDK — [build.gradle](#)

La `xray` branche de l'application inclut l'utilisation d'[annotations HTTPClient](#), de [requêtes SQL](#), de [sous-segments personnalisés](#), d'une [AWS Lambda](#) fonction instrumentée, ainsi que de codes et de scripts d'[initialisation instrumentés](#).

Pour permettre la connexion et l' AWS SDK pour JavaScript utilisation des utilisateurs dans le navigateur, la `xray-cognito` succursale ajoute Amazon Cognito afin de prendre en charge l'authentification et l'autorisation des utilisateurs. Avec les informations d'identification extraites d'Amazon Cognito, l'application Web envoie également des données de suivi à X-Ray pour enregistrer les informations relatives aux demandes du point de vue du client. Le client du navigateur apparaît sous la forme de son propre nœud sur la carte de trace et enregistre des informations supplémentaires, notamment l'URL de la page que l'utilisateur consulte et son identifiant.

Enfin, la `xray-worker` branche ajoute une fonction Lambda Python instrumentée qui s'exécute indépendamment et traite les éléments d'une file d'attente Amazon SQS. Scorekeep ajoute un élément à la file d'attente chaque fois qu'un jeu se termine. Le travailleur Lambda, déclenché par des CloudWatch événements, extrait les éléments de la file d'attente toutes les quelques minutes et les traite pour stocker les enregistrements de jeu dans Amazon S3 à des fins d'analyse.

Rubriques

- [Commencer à utiliser l'exemple d'application Scorekeep](#)
- [Instrumentation manuelle des clients du AWS SDK](#)
- [Création de sous-segments supplémentaires](#)
- [Enregistrement des annotations, des métadonnées et de l'utilisateur IDs](#)
- [Instrumentation des appels HTTP sortants](#)
- [Instrumentation d'appels vers une base de données PostgreSQL](#)
- [Fonctions d'instrumentation AWS Lambda](#)
- [Instrumentation du code de démarrage](#)
- [Instrumentation de scripts](#)
- [Instrumentation d'un client d'application web](#)
- [Utilisation de clients instrumentés dans les threads de travail](#)

Commencer à utiliser l'exemple d'application Scorekeep

Ce didacticiel utilise la `xray-gettingstarted` branche de l'[exemple d'application Scorekeep](#), qui permet de AWS CloudFormation créer et de configurer les ressources qui exécutent l'exemple

d'application et le daemon X-Ray sur Amazon ECS. L'application utilise le framework Spring pour implémenter une API Web JSON et AWS SDK pour Java pour conserver les données dans Amazon DynamoDB. Un filtre de servlet intégré à l'application gère toutes les demandes entrantes traitées par l'application, et un gestionnaire de demandes sur le client du AWS SDK gère les appels en aval à DynamoDB.

Vous pouvez suivre ce didacticiel en utilisant le AWS Management Console ou le AWS CLI.

Sections

- [Prérequis](#)
- [Installez l'application Scorekeep à l'aide de CloudFormation](#)
- [Génération de données de suivi](#)
- [Consultez la carte de suivi dans le AWS Management Console](#)
- [Configuration des notifications Amazon SNS](#)
- [Exploration de l'exemple d'application](#)
- [Facultatif : stratégie de moindre privilège](#)
- [Nettoyage](#)
- [Étapes suivantes](#)

Prérequis

Ce didacticiel permet AWS CloudFormation de créer et de configurer les ressources qui exécutent l'exemple d'application et le daemon X-Ray. Les prérequis suivants sont requis pour installer et exécuter le didacticiel :

1. Si vous utilisez un utilisateur IAM avec des autorisations limitées, ajoutez les politiques utilisateur suivantes dans la console [IAM](#) :
 - `AWSCloudFormationFullAccess`— pour accéder et utiliser CloudFormation
 - `AmazonS3FullAccess`— pour télécharger un fichier modèle à CloudFormation l'aide du AWS Management Console
 - `IAMFullAccess`— pour créer les rôles d' EC2 instance Amazon ECS et Amazon
 - `AmazonEC2FullAccess`— pour créer les EC2 ressources Amazon
 - `AmazonDynamoDBFullAccess`— pour créer les tables DynamoDB
 - `AmazonECS_FullAccess`— pour créer des ressources Amazon ECS

- `AmazonSNSFullAccess`— pour créer la rubrique Amazon SNS
 - `AWSXrayReadOnlyAccess`— pour obtenir l'autorisation de consulter la carte de suivi et les traces dans la console X-Ray
2. Pour exécuter le didacticiel à l'aide de AWS CLI, [installez la version 2.7.9 ou ultérieure de la CLI](#) et [configurez la CLI](#) avec l'utilisateur de l'étape précédente. Assurez-vous que la région est configurée lors de la configuration AWS CLI avec l'utilisateur. Si aucune région n'est configurée, vous devrez l'ajouter `--region AWS-REGION` à chaque commande de la CLI.
 3. Assurez-vous que [Git](#) est installé afin de cloner l'exemple de dépôt d'applications.
 4. Utilisez l'exemple de code suivant pour cloner la `xray-gettingstarted` branche du référentiel Scorekeep :

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

Installez l'application Scorekeep à l'aide de CloudFormation

AWS Management Console

Installez l'exemple d'application à l'aide du AWS Management Console

1. Ouvrez la [console CloudFormation](#).
2. Choisissez Créer une pile, puis choisissez Avec de nouvelles ressources dans le menu déroulant.
3. Dans la section Spécifier un modèle, sélectionnez Charger un modèle de fichier.
4. Sélectionnez Choisir un fichier, accédez au `xray-scorekeep/cloudformation` dossier créé lorsque vous avez cloné le dépôt git, puis choisissez le `cf-resources.yaml` fichier.
5. Choisissez Next (Suivant) pour continuer.
6. Entrez `scorekeep` dans la zone de texte Nom de la pile, puis choisissez Suivant au bas de la page pour continuer. Notez que le reste de ce didacticiel suppose que la pile est nommée `scorekeep`.
7. Faites défiler la page des options de configuration de la pile vers le bas et choisissez Suivant pour continuer.

8. Faites défiler la page de révision vers le bas, cochez la case indiquant que des ressources IAM CloudFormation peuvent être créées avec des noms personnalisés, puis choisissez Create stack.
9. La CloudFormation pile est en cours de création. L'état de la pile CREATE_IN_PROGRESS durera environ cinq minutes avant de passer à CREATE_COMPLETE. Le statut sera actualisé régulièrement, ou vous pouvez actualiser la page.

AWS CLI

Installez l'exemple d'application à l'aide du AWS CLI

1. Accédez au cloudformation dossier du xray-scorekeep référentiel que vous avez cloné plus tôt dans le didacticiel :

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

2. Entrez la AWS CLI commande suivante pour créer la CloudFormation pile :

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

3. Attendez que l'état de la CloudFormation pile soit CREATE_COMPLETE rétabli, ce qui prendra environ cinq minutes. Utilisez la AWS CLI commande suivante pour vérifier l'état :

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

Génération de données de suivi

L'exemple d'application inclut une application web frontale. Utilisez l'application Web pour générer du trafic vers l'API et envoyer des données de suivi à X-Ray. Tout d'abord, récupérez l'URL de l'application Web à l'aide du AWS Management Console ou du AWS CLI :

AWS Management Console

Trouvez l'URL de l'application à l'aide du AWS Management Console

1. Ouvrez la [console CloudFormation](#).

2. Choisissez la scorekeep pile dans la liste.
3. Choisissez l'onglet Sorties sur la page de scorekeep pile, puis choisissez le lien LoadBalancerUrl URL pour ouvrir l'application Web.

AWS CLI

Trouvez l'URL de l'application à l'aide du AWS CLI

1. Utilisez la commande suivante pour afficher l'URL de l'application Web :

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

2. Copiez cette URL et ouvrez-la dans un navigateur pour afficher l'application Web Scorekeep.

Utiliser l'application Web pour générer des données de suivi

1. Choisissez Create pour créer un utilisateur et une session.
2. Saisissez un nom de jeu, définissez les règles sur Tic Tac Toe, puis choisissez Create pour créer un jeu.
3. Choisissez Play pour lancer le jeu.
4. Choisissez une case pour jouer et modifier l'état du jeu.

Chacune de ces étapes génère des requêtes HTTP adressées à l'API et des appels en aval à DynamoDB pour lire et écrire les données relatives à l'utilisateur, à la session, au jeu, au déplacement et à l'état.

Consultez la carte de suivi dans le AWS Management Console

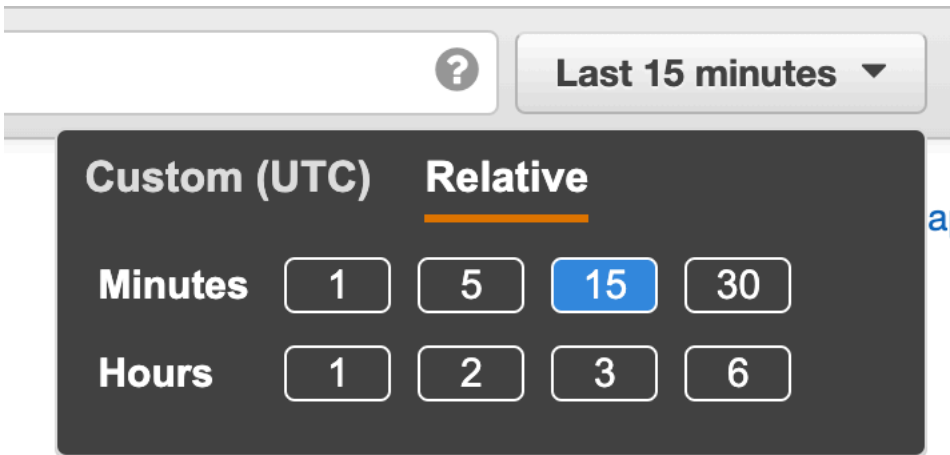
Vous pouvez consulter la carte de suivi et les traces générées par l'exemple d'application dans X-Ray et CloudWatch les consoles.

X-Ray console

Utiliser la console X-Ray

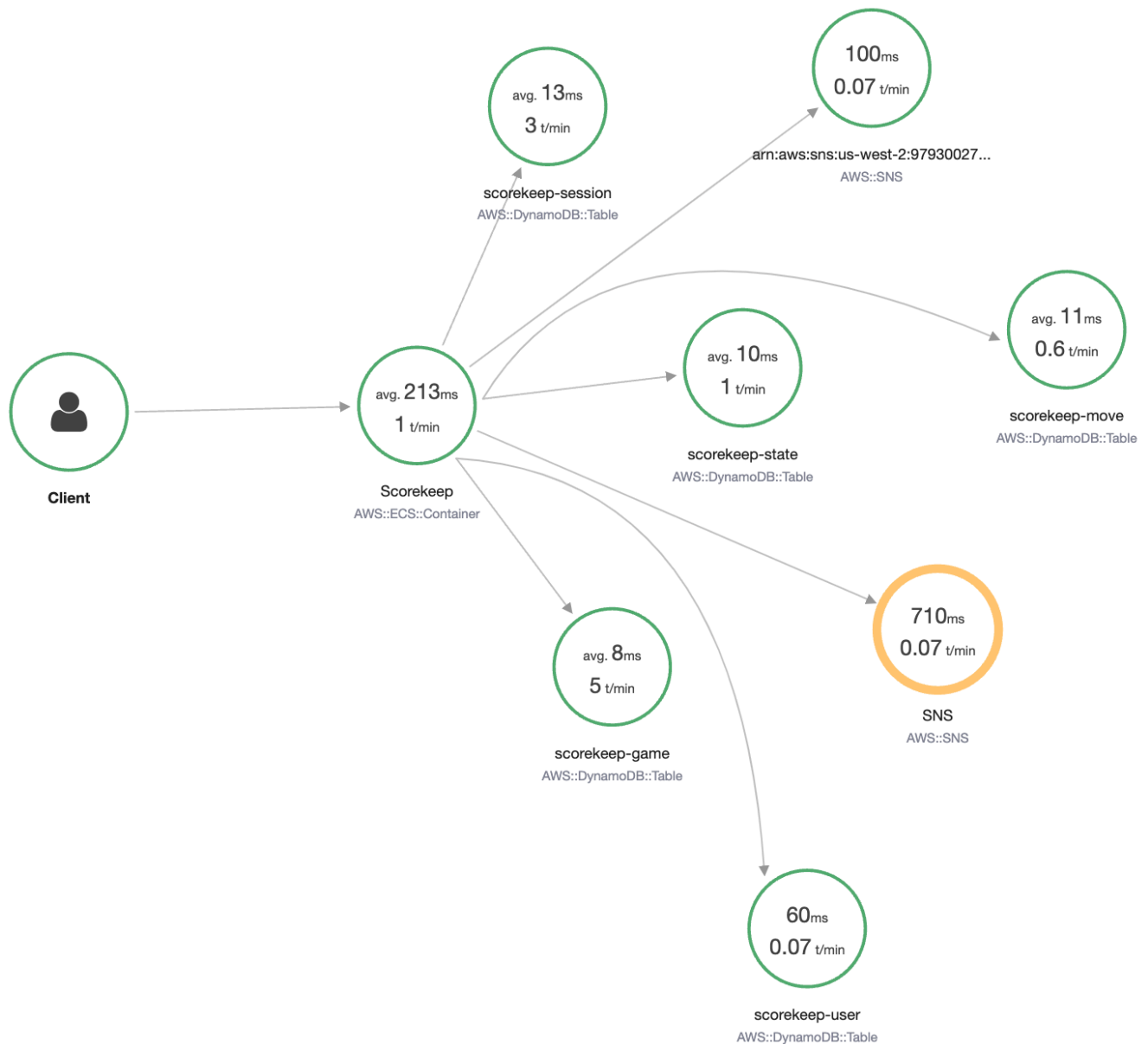
1. Ouvrez la page Trace Map de la [console X-Ray](#).

2. La console affiche une représentation du graphe de service généré par X-Ray à partir des données de suivi envoyées par l'application. Assurez-vous d'ajuster la période de la carte de trace si nécessaire, afin de vous assurer qu'elle affichera toutes les traces depuis le premier démarrage de l'application Web.



La carte de trace montre le client de l'application Web, l'API exécutée dans Amazon ECS et chaque table DynamoDB utilisée par l'application. Chaque demande à l'application, jusqu'à un nombre maximal de demandes configurables par seconde, est suivi lorsqu'il atteint l'API, génère des demandes aux services en aval, et se termine.

Vous pouvez choisir n'importe quel nœud dans le graphique de services pour afficher les suivis des demandes ayant généré du trafic vers ce nœud. Actuellement, le nœud Amazon SNS est jaune. Explorez ce nœud pour savoir pourquoi.



Pour trouver la cause de l'erreur

1. Choisissez le nœud nommé SNS. Le panneau des détails du nœud s'affiche.
2. Choisissez View traces (Afficher les suivis) pour accéder à l'écran Trace overview (Présentation des suivis).
3. Choisissez le suivi depuis la Trace list (Liste des suivis). Ce suivi n'est pas associé à une méthode ou une URL, car il a été enregistré au démarrage et non en réponse à une demande entrante.

Q service("SNS") ? Last 5 Minutes ↺

Trace overview

Group by: URL

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. Cliquez sur l'icône d'état d'erreur dans le segment Amazon SNS au bas de la page, pour ouvrir la page Exceptions du sous-segment SNS.

[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

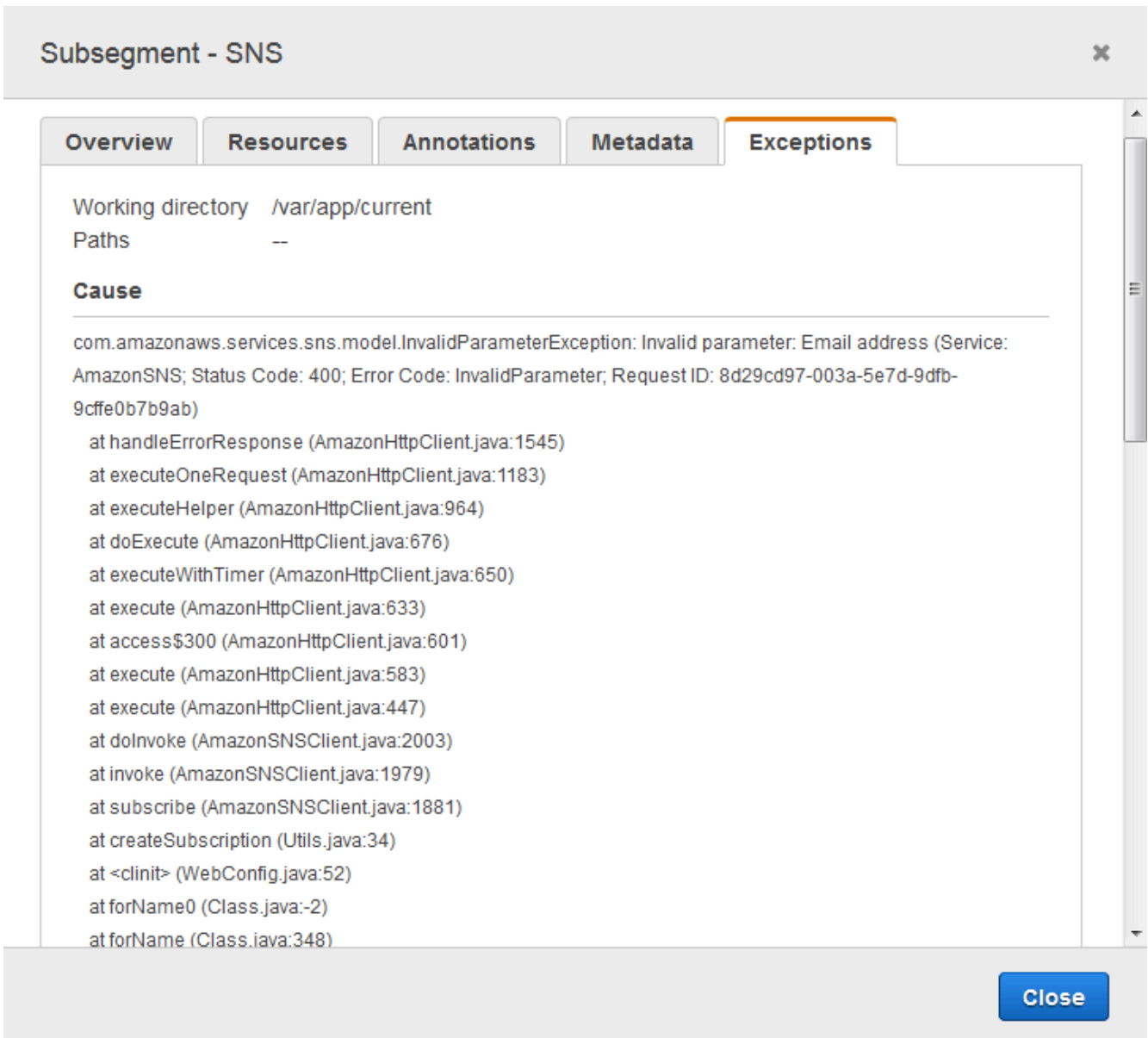
```

graph LR
    Client((Client)) --> Scorekeep((2.11s  
1 Request  
Scorekeep  
AWS::EC2::Instance))
    Scorekeep --> SNS((728ms  
1 Request  
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
▼ Scorekeep AWS::EC2::Instance															
Scorekeep	-	2.1 sec	✓	[Timeline bar for Scorekeep]											
SNS	400	728 ms	⚠	[Timeline bar for SNS]											
▶ SNS AWS::SNS (Client Response)															

5. Le SDK X-Ray capture automatiquement les exceptions émises par les clients du AWS SDK instrumentés et enregistre la trace de la pile.

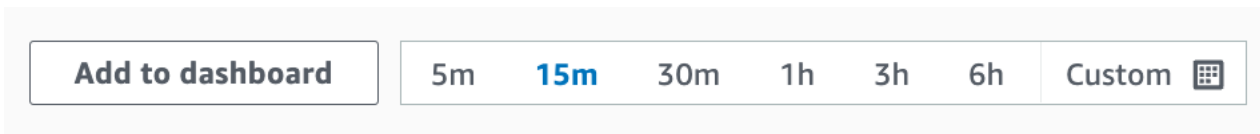


CloudWatch console

Utiliser la CloudWatch console

1. Ouvrez la page de [carte de trace X-Ray](#) de la CloudWatch console.
2. La console affiche une représentation du graphe de service généré par X-Ray à partir des données de suivi envoyées par l'application. Assurez-vous d'ajuster la période de la carte de

trace si nécessaire, afin de vous assurer qu'elle affichera toutes les traces depuis le premier démarrage de l'application Web.



La carte de trace montre le client de l'application Web, l'API exécutée dans Amazon EC2 et chaque table DynamoDB utilisée par l'application. Chaque demande à l'application, jusqu'à un nombre maximal de demandes configurables par seconde, est suivi lorsqu'il atteint l'API, génère des demandes aux services en aval, et se termine.

Vous pouvez choisir n'importe quel nœud dans le graphique de services pour afficher les suivis des demandes ayant généré du trafic vers ce nœud. Actuellement, le nœud Amazon SNS est orange. Explorez ce nœud pour savoir pourquoi.



Pour trouver la cause de l'erreur

1. Choisissez le nœud nommé SNS. Le panneau de détails du nœud SNS s'affiche sous la carte.
2. Choisissez Afficher les traces pour accéder à la page Traces.
3. Ajoutez le bas de la page, choisissez la trace dans la liste des traces. Ce suivi n'est pas associé à une méthode ou une URL, car il a été enregistré au démarrage et non en réponse à une demande entrante.

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](#).

[Run query](#) ✓ 1 traces retrieved

[▶](#) **Query refiners**

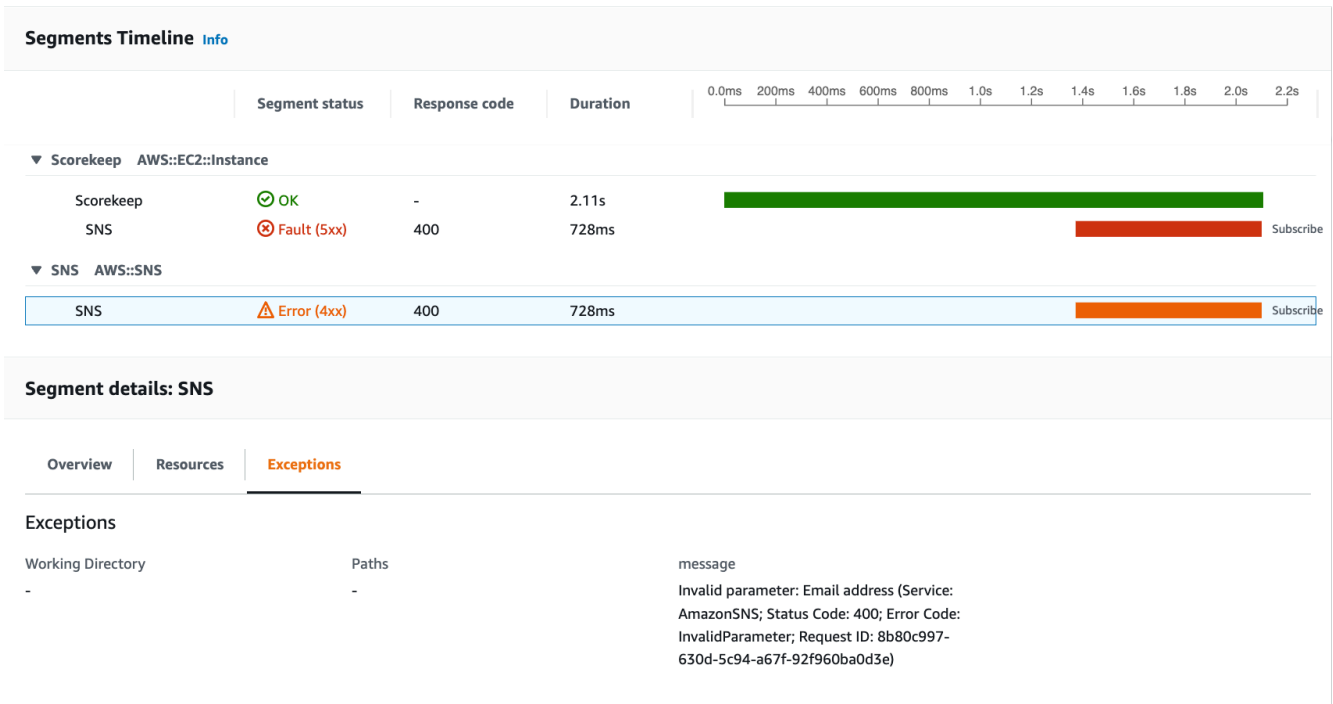
Traces (1) [Add to dashboard](#)

This table shows the most recent traces with an average response time of 2.11s. It shows as many as 1000 traces.

< 1 > [⚙️](#)

ID	Trace status	Timestamp	Response code	Response Time	Duration
...86b347fc50bc57a992e9b835	✓ OK	19.1min (2022-08-10 12:05:25)	-	2.11s	2.11s

4. Choisissez le sous-segment Amazon SNS en bas de la chronologie des segments, puis cliquez sur l'onglet Exceptions correspondant au sous-segment SNS pour afficher les détails des exceptions.



La cause indique que l'adresse e-mail fournie dans un appel vers `createSubscription` effectué dans la classe `WebConfig` n'était pas valide. Dans la section suivante, nous allons résoudre ce problème.

Configuration des notifications Amazon SNS

Scorekeep utilise Amazon SNS pour envoyer des notifications lorsque les utilisateurs terminent un jeu. Lorsque l'application démarre, elle essaie de créer un abonnement pour une adresse e-mail définie dans un paramètre de CloudFormation pile. Cet appel échoue actuellement. Configurez un e-mail de notification pour activer les notifications et résoudre les défaillances mises en évidence sur la carte de suivi.

AWS Management Console

Pour configurer les notifications Amazon SNS à l'aide du AWS Management Console

1. Ouvrez la [console CloudFormation](#).
2. Cliquez sur le bouton radio situé à côté du nom de la scorekeep pile dans la liste, puis sélectionnez Mettre à jour.
3. Assurez-vous que l'option Utiliser le modèle actuel est sélectionnée, puis cliquez sur Suivant sur la page Mettre à jour la pile.

- Recherchez le paramètre Email dans la liste et remplacez la valeur par défaut par une adresse e-mail valide.

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

- Faites défiler la page vers le bas, puis choisissez Next (Suivant).
- Faites défiler la page de révision vers le bas, cochez la case indiquant que des ressources IAM CloudFormation peuvent être créées avec des noms personnalisés, puis choisissez Update stack.
- La CloudFormation pile est en cours de mise à jour. L'état de la pile UPDATE_IN_PROGRESS durera environ cinq minutes avant de passer àUPDATE_COMPLETE. Le statut sera actualisé régulièrement, ou vous pouvez actualiser la page.

AWS CLI

Pour configurer les notifications Amazon SNS à l'aide du AWS CLI

- Accédez au xray-scorekeep/cloudformation/ dossier que vous avez créé précédemment et ouvrez le cf-resources.yaml fichier dans un éditeur de texte.
- Recherchez la Default valeur dans le paramètre Email et remplacez-la par **UPDATE_ME** une adresse e-mail valide.

Parameters:**Email:**

Type: String

Default: UPDATE_ME # <- change to a valid abc@def.xyz email address

- Dans le cloudformation dossier, mettez à jour la CloudFormation pile à l'aide de la AWS CLI commande suivante :

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

4. Attendez que l'état de la CloudFormation pile soit UPDATE_COMPLETE rétabli, ce qui prendra quelques minutes. Utilisez la AWS CLI commande suivante pour vérifier l'état :

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

Lorsque la mise à jour est terminée, Scorekeep redémarre et crée un abonnement à la rubrique SNS. Vérifiez votre adresse e-mail et confirmez l'abonnement pour voir les mises à jour lorsque vous terminez un jeu. Ouvrez la carte de suivi pour vérifier que les appels à SNS n'échouent plus.

Exploration de l'exemple d'application

L'exemple d'application est une API Web HTTP en Java configurée pour utiliser le SDK X-Ray pour Java. Lorsque vous déployez l'application avec le CloudFormation modèle, elle crée les tables DynamoDB, le cluster Amazon ECS et les autres services nécessaires pour exécuter Scorekeep sur ECS. Un fichier de définition de tâche pour ECS est créé via CloudFormation. Ce fichier définit les images de conteneur utilisées par tâche dans un cluster ECS. Ces images sont obtenues auprès de l'ECR public officiel de X-Ray. L'image du conteneur de l'API scorekeep contient l'API compilée avec Gradle. L'image de conteneur du conteneur frontal Scorekeep sert le frontend à l'aide du serveur proxy nginx. Ce serveur achemine les demandes vers des chemins commençant par /api vers l'API.

Pour instrumenter les demandes HTTP, l'application ajoute le `TracingFilter` fourni par le kit de développement logiciel.

Exemple `src/main/java/scorekeep/WebConfig.java` - filtre de servlet

```
import javax.servlet.Filter;  
import com.amazonaws.xray.java.servlet.AWSXRayServletFilter;  
...  
  
@Configuration  
public class WebConfig {  
  
    @Bean  
    public Filter TracingFilter() {  
        return new AWSXRayServletFilter("Scorekeep");  
    }  
    ...  
}
```

Ce filtre envoie des données de suivi sur toutes les demandes entrantes servies par l'application, y compris l'URL de demande, la méthode, le statut de réponse, l'heure de début et de fin.

L'application effectue également des appels en aval à DynamoDB à l'aide du AWS SDK pour Java. Pour instrumenter ces appels, l'application utilise simplement les sous-modules AWS liés au SDK comme dépendances, et le SDK X-Ray pour Java instrumente automatiquement tous les clients du SDK. AWS

L'application utilise Docker pour créer le code source sur instance avec le fichier Gradle Docker Image et le Scorekeep API Dockerfile fichier pour exécuter le fichier JAR exécutable généré par Gradle. ENTRYPOINT

Exemple utilisation de Docker pour créer via Gradle Docker Image

```
docker run --rm -v /PATH/TO/SCOREKEEP_REPO/home/gradle/project -w /home/gradle/project
gradle:4.3 gradle build
```

Exemple POINT D'ENTRÉE Dockerfile

```
ENTRYPOINT [ "sh", "-c", "java -Dserver.port=5000 -jar scorekeep-api-1.0.0.jar" ]
```

Le fichier build.gradle télécharge les sous-modules du kit de développement logiciel à partir de Maven pendant la compilation en les déclarant comme des dépendances.

Exemple build.gradle - dépendances

```
...
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")
    }
}
```

```
}
```

Les sous-modules principaux, AWS SDK et AWS SDK Instrumentor sont tout ce qui est nécessaire pour instrumenter automatiquement tous les appels en aval effectués avec le SDK. AWS

Pour relayer les données de segment brutes vers l'API X-Ray, le daemon X-Ray doit écouter le trafic sur le port UDP 2000. Pour ce faire, l'application fait exécuter le daemon X-Ray dans un conteneur qui est déployé parallèlement à l'application Scorekeep sur ECS en tant que conteneur annexe. Consultez la rubrique consacrée au [daemon X-Ray](#) pour plus d'informations.

Exemple Définition du conteneur X-Ray Daemon dans une définition de tâche ECS

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

Le SDK X-Ray pour Java fournit une classe `AWSXRay` nommée qui fournit l'enregistreur global, `TracingHandler` que vous pouvez utiliser pour instrumenter votre code. Vous pouvez configurer l'enregistreur mondial pour personnaliser le `AWSXRayServletFilter` qui crée des segments pour les appels HTTP entrants. L'exemple inclut un bloc statique dans la classe `WebConfig` qui configure l'enregistreur mondial avec des plug-ins et des règles d'échantillonnage.

Exemple `src/main/java/scorekeep/WebConfig.java` - enregistreur

```
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());
        ...
    }
}
```

Cet exemple utilise le générateur pour charger des règles d'échantillonnage à partir d'un fichier nommé `sampling-rules.json`. Les règles d'échantillonnage déterminent le taux auquel le kit SDK enregistre des segments pour les demandes entrantes.

Exemple `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
}
}

```

Le fichier de règles d'échantillonnage définit quatre règles personnalisées et une règle par défaut. Pour chaque demande entrante, le kit SDK évalue les règles personnalisées dans l'ordre dans lequel elles sont définies. Le kit de développement logiciel (SDK) applique la première règle qui correspond à la méthode, au chemin et au nom de service de la demande. Pour Scorekeep, la première règle intercepte toutes les demandes POST (appels de création de ressource) en appliquant un objectif fixé à une demande par seconde et un taux de 1,0 ou 100 % des demandes une fois l'objectif fixé atteint.

Les trois autres règles personnalisées appliquent un taux de 5 %, sans aucun objectif fixé concernant les lectures de session, de jeu et d'état (demandes GET). Il s'ensuit une réduction du nombre de suivis pour les appels périodiques émis automatiquement par le service frontal à quelques secondes d'intervalle pour vérifier que le contenu est à jour. Pour toutes les autres demandes, le fichier définit un taux par défaut d'une demande par seconde et un taux de 10 %.

L'exemple d'application indique également comment utiliser des fonctionnalités avancées, telles que l'instrumentation manuelle d'un client de kit de développement logiciel, la création de sous-segments

supplémentaires et des appels HTTP sortants. Pour de plus amples informations, veuillez consulter [AWS X-Ray exemple d'application](#).

Facultatif : stratégie de moindre privilège

Les conteneurs Scorekeep ECS accèdent aux ressources en utilisant des politiques d'accès complet, telles que `AmazonSNSFullAccess` et `AmazonDynamoDBFullAccess`. L'utilisation de politiques d'accès complet n'est pas la meilleure pratique pour les applications de production. L'exemple suivant met à jour la politique DynamoDB IAM afin d'améliorer la sécurité de l'application. Pour en savoir plus sur les meilleures pratiques de sécurité dans les politiques IAM, consultez la section [Gestion des identités et des accès pour AWS X-Ray](#).

Exemple Modèle cf-resources.yaml Définition du rôle ECSTask

```
ECSTaskRole:
  Type: AWS::IAM::Role
  Properties:
    AssumeRolePolicyDocument:
      Version: "2012-10-17"
      Statement:
        -
          Effect: "Allow"
          Principal:
            Service:
              - "ecs-tasks.amazonaws.com"
          Action:
            - "sts:AssumeRole"
    ManagedPolicyArns:
      - "arn:aws:iam::aws:policy/AmazonDynamoDBFullAccess"
      - "arn:aws:iam::aws:policy/AmazonSNSFullAccess"
      - "arn:aws:iam::aws:policy/AWSXrayFullAccess"
    RoleName: "scorekeepRole"
```

Pour mettre à jour votre politique, vous devez d'abord identifier l'ARN de vos ressources DynamoDB. Vous utilisez ensuite l'ARN dans une politique IAM personnalisée. Enfin, vous devez appliquer cette politique à votre profil d'instance.

Pour identifier l'ARN de votre ressource DynamoDB :

1. Ouvrez la [console DynamoDB](#).
2. Choisissez Tables dans la barre de navigation de gauche.

3. Choisissez l'une des options `scorekeep-*` pour afficher la page détaillée du tableau.
4. Dans l'onglet Vue d'ensemble, choisissez Informations supplémentaires pour développer la section et afficher le nom de la ressource Amazon (ARN). Copiez cette valeur.
5. Insérez l'ARN dans la politique IAM suivante, en remplaçant les `AWS_ACCOUNT_ID` valeurs `AWS_REGION` et par votre région et votre identifiant de compte spécifiques. Cette nouvelle politique n'autorise que les actions spécifiées, au lieu de la `AmazonDynamoDBFullAccess` politique qui autorise n'importe quelle action.

Exemple

```
{
  "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-*"
    }
  ]
}
```

Les tables créées par l'application suivent une convention de dénomination cohérente. Vous pouvez utiliser ce `scorekeep-*` format pour indiquer tous les tableaux Scorekeep.

Modifiez votre politique IAM

1. Ouvrez le [rôle de tâche Scorekeep \(ScorekeepRole\) depuis la console IAM](#).
2. Cochez la case à côté de la `AmazonDynamoDBFullAccess` politique et choisissez Supprimer pour supprimer cette politique.
3. Choisissez Ajouter des autorisations, puis Joindre des politiques, et enfin Créer une politique.

4. Choisissez l'onglet JSON et collez la politique créée ci-dessus.
5. Choisissez Next : Tags au bas de la page.
6. Choisissez Suivant : Révision au bas de la page.
7. Pour Nom, attribuez un nom à la politique.
8. Choisissez Créer une politique au bas de la page.
9. Attachez la politique nouvellement créée au `scorekeepRole` rôle. L'entrée en vigueur de la politique ci-jointe peut prendre quelques minutes.

Si vous avez attaché la nouvelle politique au `scorekeepRole` rôle, vous devez la détacher avant de supprimer la CloudFormation pile, car cette politique attachée empêchera la suppression de la pile. La politique peut être automatiquement détachée en la supprimant.

Supprimer votre politique IAM personnalisée

1. Ouvrez la [console IAM](#).
2. Choisissez Politiques dans la barre de navigation de gauche.
3. Recherchez le nom de stratégie personnalisée que vous avez créé plus tôt dans cette section, puis cliquez sur le bouton radio à côté du nom de la stratégie pour le mettre en évidence.
4. Choisissez le menu déroulant Actions, puis sélectionnez Supprimer.
5. Tapez le nom de la politique personnalisée, puis choisissez Supprimer pour confirmer la suppression. Cela détachera automatiquement la politique du `scorekeepRole` rôle.

Nettoyage

Pour supprimer les ressources de l'application Scorekeep, procédez comme suit :

Note

Si vous avez créé et joint des politiques personnalisées à l'aide de la section précédente de ce didacticiel, vous devez supprimer la politique de la pile `scorekeepRole` avant de supprimer la CloudFormation pile.

AWS Management Console

Supprimez l'exemple d'application à l'aide du AWS Management Console

1. Ouvrez la [console CloudFormation](#).
2. Cliquez sur le bouton radio situé à côté du nom de la `scorekeep` pile dans la liste, puis sélectionnez Supprimer.
3. La CloudFormation pile est en train d'être supprimée. L'état de la pile `DELETE_IN_PROGRESS` durera quelques minutes jusqu'à ce que toutes les ressources soient supprimées. Le statut sera actualisé régulièrement, ou vous pouvez actualiser la page.

AWS CLI

Supprimez l'exemple d'application à l'aide du AWS CLI

1. Entrez la AWS CLI commande suivante pour supprimer la CloudFormation pile :

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

2. Attendez que la CloudFormation pile n'existe plus, ce qui prendra environ cinq minutes. Utilisez la AWS CLI commande suivante pour vérifier l'état :

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

Étapes suivantes

Apprenez-en plus sur X-Ray dans le chapitre suivant, [AWS X-Ray concepts](#).

Pour instrumenter votre propre application, apprenez-en plus sur le SDK X-Ray pour Java ou sur l'un des autres logiciels de X-Ray SDKs :

- SDK X-Ray pour Java — [AWS X-Ray SDK pour Java](#)
- Kit de développement X-Ray pour Node.js — [AWS Kit de développement X-Ray pour Node.js](#)
- Kit de développement X-Ray pour .NET — [AWS X-Ray SDK pour .NET](#)

Pour exécuter le daemon X-Ray en local ou en mode actif AWS, consultez [AWS X-Ray démon](#).

Pour contribuer à l'exemple d'application sur GitHub, voir [eb-java-scorekeep](#).

Instrumentation manuelle des clients du AWS SDK

Le SDK X-Ray pour Java instrumente automatiquement AWS tous les clients du SDK lorsque [vous incluez AWS le sous-module SDK Instrumentor](#) dans vos dépendances de build.

Vous pouvez désactiver l'instrumentation automatique du client en supprimant le sous-module Instrumentor. Ainsi, vous pourrez instrumenter certains clients manuellement tout en ignorant les autres, ou utiliser différents gestionnaires de suivi sur différents clients.

Pour illustrer la prise en charge de l'instrumentation de clients AWS SDK spécifiques, l'application transmet un gestionnaire de suivi en `AmazonDynamoDBClientBuilder` tant que gestionnaire de demandes dans le modèle utilisateur, jeu et session. Cette modification de code indique au SDK d'instrumenter tous les appels à DynamoDB à l'aide de ces clients.

Exemple [src/main/java/scorekeep/SessionModel.java](#)— Instrumentation client manuelle du AWS SDK

```
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);
```

Si vous supprimez le sous-module AWS SDK Instrumentor des dépendances du projet, seuls les clients du AWS SDK instrumentés manuellement apparaissent dans la carte de trace.

Création de sous-segments supplémentaires

Dans la classe des modèles d'utilisateur, l'application crée manuellement les sous-segments pour regrouper tous les appels en aval effectués au sein de la fonction `saveUser` et ajoute les métadonnées.

Exemple [src/main/java/scorekeep/UserModel.java](#) - Sous-segments personnalisés

```
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();
    }
}
```

Enregistrement des annotations, des métadonnées et de l'utilisateur IDs

Dans la classe de modèle de jeu, l'application enregistre les Game objets dans un bloc de [métadonnées](#) chaque fois qu'elle enregistre un jeu dans DynamoDB. Séparément, l'application enregistre le jeu IDs dans [des annotations destinées](#) à être utilisées avec des [expressions de filtre](#).

Exemple [src/main/java/scorekeep/GameModel.java](#)— Annotations et métadonnées

```
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();
    }
}

```

Dans le contrôleur de déplacement, l'application enregistre [l'utilisateur IDs](#) avec `setUser`. IDs Les utilisateurs sont enregistrés dans un champ distinct sur les segments et sont indexés pour être utilisés dans le cadre de la recherche.

Exemple [src/main/java/scorekeep/MoveController.java](#) — ID utilisateur

```

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);
}

```

Instrumentation des appels HTTP sortants

La classe `UserFactory` montre comment l'application utilise la version du SDK X-Ray pour Java pour `HttpClientBuilder` instrumenter les appels HTTP sortants.

Exemple [src/main/java/scorekeep/UserFactory.java](#) — `HttpClient` instrumentation

```

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();
}
}
```

Si vous utilisez actuellement `org.apache.http.impl.client.HttpClientBuilder`, vous pouvez simplement remplacer l'instruction d'importation par cette classe avec une pour `com.amazonaws.xray.proxies.apache.http.HttpClientBuilder`.

Instrumentation d'appels vers une base de données PostgreSQL

Le `application-pgsql.properties` fichier ajoute l'intercepteur de traçage X-Ray PostgreSQL à la source de données créée dans [RdsWebConfig.java](#)

Exemple [application-pgsql.properties](#)— Instrumentation de base de données PostgreSQL

```
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

Consultez [Configuration des bases de données avec Elastic Beanstalk](#) dans le Manuel du développeur AWS Elastic Beanstalk pour obtenir des détails sur l'ajout d'une base de données PostgreSQL à l'environnement de l'application.

La page de démonstration de X-Ray de la `xray` branche inclut une démonstration qui utilise la source de données instrumentée pour générer des traces contenant des informations sur les requêtes SQL qu'elle génère. Accédez au chemin `/#/xray` dans l'application en cours d'exécution ou choisissez **Powered by AWS X-Ray** dans la barre de navigation pour afficher la page de démonstration.

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.

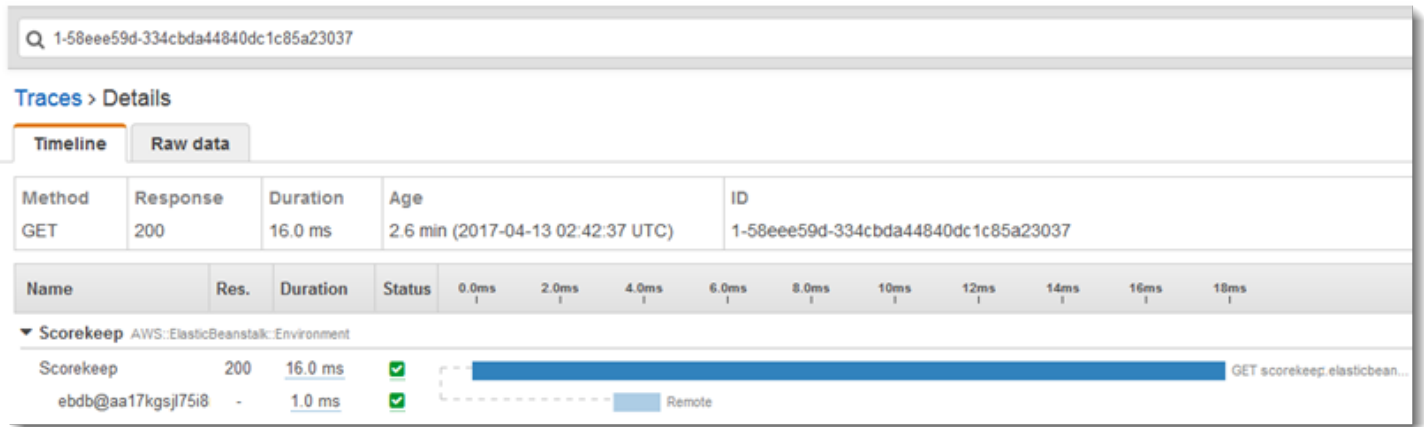
Trace SQL queries

[View traces in AWS X-Ray](#)

ID	Winner	Loser
1	Mugur	Gheorghită
2	Paula	Adorján
3	Αρχίας	Stela
4	付	Pərvanə

Choisissez Trace SQL queries pour simuler les sessions de jeu et stocker les résultats dans la base de données attachée. Choisissez ensuite View traces in AWS X-Ray pour afficher une liste filtrée des traces qui suivent l'/api/historyitinaire de l'API.

Choisissez l'un des suivis de la liste pour afficher la chronologie, requête SQL incluse.



Fonctions d'instrumentation AWS Lambda

Scorekeep utilise deux AWS Lambda fonctions. La première est une fonction Node.js de la branche `lambda` qui génère des noms aléatoires pour les nouveaux utilisateurs. Lorsqu'un utilisateur crée une session sans indiquer de nom, l'application appelle une fonction nommée `random-name` avec le kit AWS SDK pour Java. Le SDK X-Ray pour Java enregistre les informations relatives à l'appel à Lambda dans un sous-segment comme tout autre appel effectué avec un client du SDK instrumenté. AWS

Note

L'exécution de la fonction `random-name` Lambda nécessite la création de ressources supplémentaires en dehors de l'environnement Elastic Beanstalk. Consultez le fichier `readme` pour plus d'informations et d'instructions : Intégration [AWS Lambda](#).

La deuxième fonction, `scorekeep-worker`, est une fonction Python qui s'exécute indépendamment de l'API du Scorekeep. Lorsqu'un jeu se termine, l'API enregistre l'ID de session et l'ID de jeu dans une file d'attente SQS. La fonction de travail lit les éléments de la file d'attente et appelle l'API Scorekeep pour créer des enregistrements complets de chaque session de jeu à stocker dans Amazon S3.

Scorekeep inclut des AWS CloudFormation modèles et des scripts pour créer les deux fonctions. Comme vous devez associer le SDK X-Ray au code de fonction, les modèles créent les fonctions sans aucun code. Lorsque vous déployez Scorekeep, un fichier de configuration inclus dans le dossier `.ebextensions` crée un bundle source qui inclut le kit SDK et met à jour le code de fonction et la configuration avec l' AWS Command Line Interface.

Fonctions

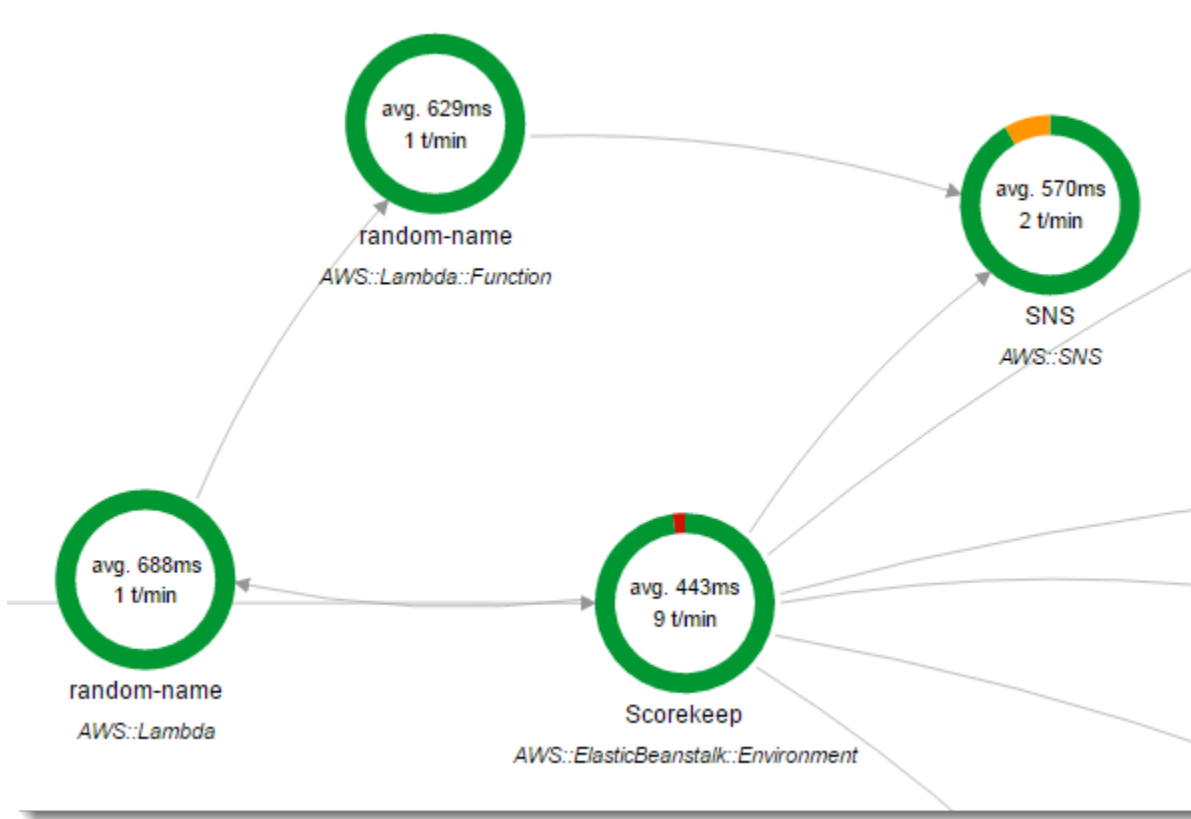
- [Nom aléatoire](#)
- [Nœuds](#)

Nom aléatoire

Scorekeep appelle la fonction de nom aléatoire lorsqu'un utilisateur lance une session de jeu sans s'être connecté ni avoir spécifié de nom d'utilisateur. Lorsque Lambda traite l'appel à `random-name`, il lit l'[en-tête de suivi](#), qui contient l'ID de trace et la décision d'échantillonnage rédigés par le SDK X-Ray pour Java.

Pour chaque requête échantillonnée, Lambda exécute le daemon X-Ray et écrit deux segments. Le premier segment enregistre les informations relatives à l'appel à Lambda qui appelle la fonction. Ce segment contient les mêmes informations que le sous-segment enregistré par Scorekeep, mais du point de vue Lambda. Le deuxième segment représente la tâche effectuée par la fonction elle-même.

Lambda transmet le segment de fonction au SDK X-Ray via le contexte de la fonction. Lorsque vous instrumentez une fonction Lambda, vous n'utilisez pas le SDK pour [créer un segment pour](#) les demandes entrantes. Lambda fournit le segment, et vous utilisez le SDK pour instrumenter les clients et écrire des sous-segments.



La fonction `random-name` est implémentée dans Node.js. Il utilise le SDK pour Node.js pour JavaScript envoyer des notifications avec Amazon SNS, et le SDK X-Ray pour Node.js pour AWS instrumenter le client du SDK. Pour écrire des annotations, la fonction crée un sous-segment personnalisé avec `AWSXRay.captureFunc`, et écrit des annotations dans la fonction instrumentée. Dans Lambda, vous ne pouvez pas écrire d'annotations directement dans le segment de fonction, mais uniquement dans un sous-segment que vous créez.

Exemple [function/index.js](#) -- Fonction Lambda de nom aléatoire

```
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;

```

Cette fonction est créée automatiquement lorsque vous déployez l'exemple d'application pour Elastic Beanstalk. La xray branche inclut un script permettant de créer une fonction Lambda vide. Les fichiers de configuration contenus dans le `.ebextensions` dossier créent le package de fonctions npm `install` pendant le déploiement, puis mettent à jour la fonction Lambda avec la CLI AWS .

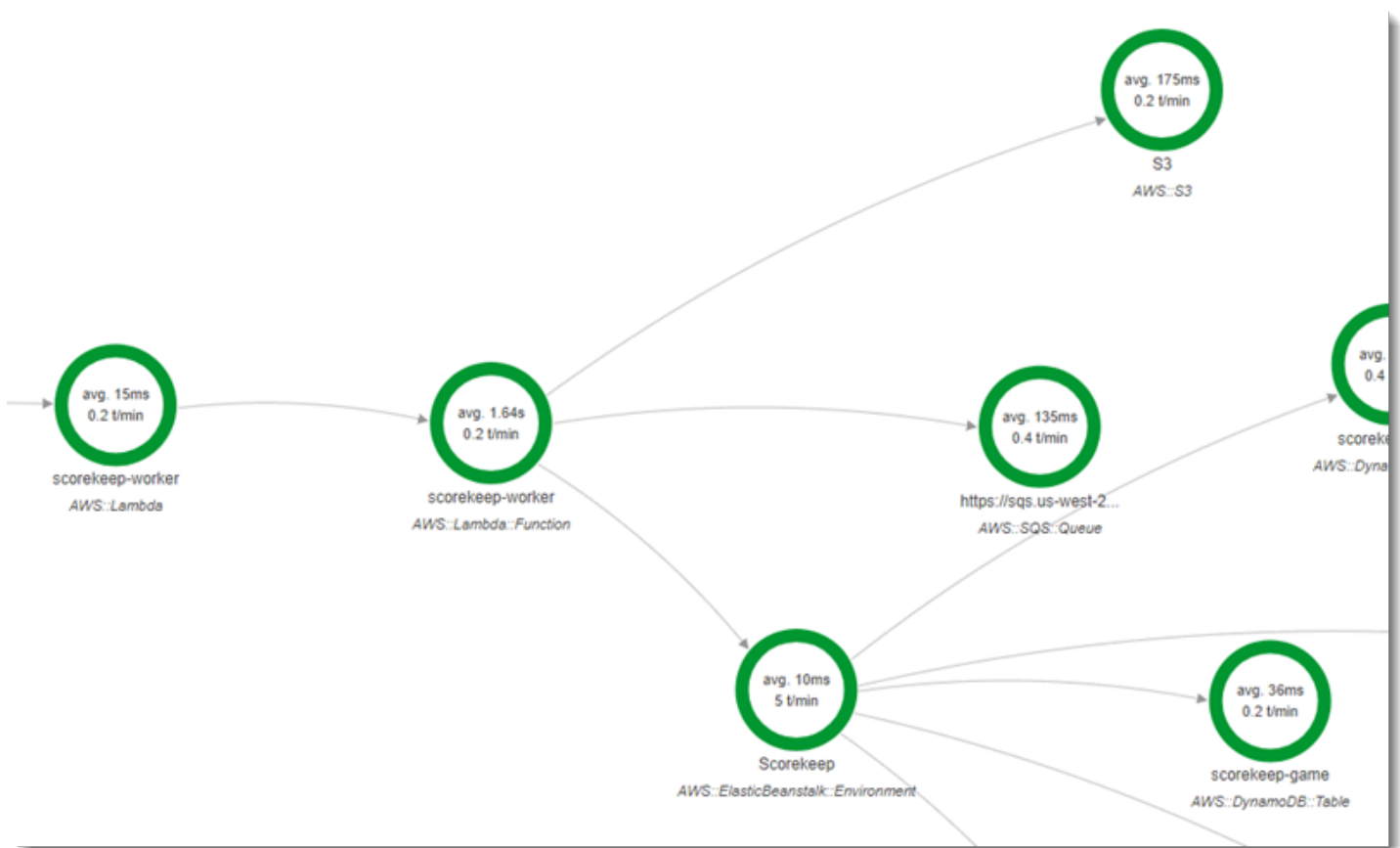
Nœuds

La fonction de travail instrumentée est fournie dans sa propre branche, `xray-worker` car elle ne peut s'exécuter que si vous créez d'abord la fonction de travail et les ressources connexes. Consultez [le fichier readme de la branche](#) pour obtenir des instructions.

La fonction est déclenchée par un événement Amazon CloudWatch Events groupé toutes les 5 minutes. Lorsqu'elle s'exécute, la fonction extrait un élément d'une file d'attente Amazon SQS gérée par Scorekeep. Chaque message contient des informations sur un jeu terminé.

Le travail extrait l'enregistrement et les documents du jeu des autres tables auxquelles cet enregistrement fait référence. Par exemple, l'enregistrement de jeu dans DynamoDB inclut une liste des mouvements exécutés pendant le jeu. La liste ne contient pas les mouvements eux-mêmes, mais plutôt IDs les mouvements enregistrés dans une table séparée.

Les sessions et les états sont également stockés en tant que références. Cela évite que les entrées conservées dans la table du jeu soient trop volumineuses, mais nécessite des appels supplémentaires pour obtenir toutes les informations sur le jeu. Le travailleur déréférence toutes ces entrées et construit un enregistrement complet du jeu sous forme de document unique dans Amazon S3. Lorsque vous souhaitez effectuer des analyses sur les données, vous pouvez exécuter des requêtes directement dans Amazon S3 avec Amazon Athena sans effectuer de migrations de données fastidieuses pour extraire vos données de DynamoDB.



Le suivi actif est activé dans la configuration de la fonction de travail, dans AWS Lambda. Contrairement à la fonction de nom aléatoire, le travailleur ne reçoit pas de demande d'une application instrumentée et AWS Lambda ne reçoit donc pas d'en-tête de suivi. Avec le traçage actif, Lambda crée l'ID de trace et prend les décisions d'échantillonnage.

Le SDK X-Ray pour Python se trouve juste à quelques lignes en haut de la fonction qui importe le SDK et exécute sa `patch_all` fonction pour le corriger AWS SDK for Python (Boto) et HTTPclients qu'il utilise pour appeler Amazon SQS et Amazon S3. Lorsque le travail exécute l'API Scorekeep, le SDK ajoute l'[en-tête de suivi](#) à la demande pour suivre les appels via l'API.

Exemple [_lambda/scorekeep-worker/scorekeep-worker.py](#) - Fonction Lambda de travail

```
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')

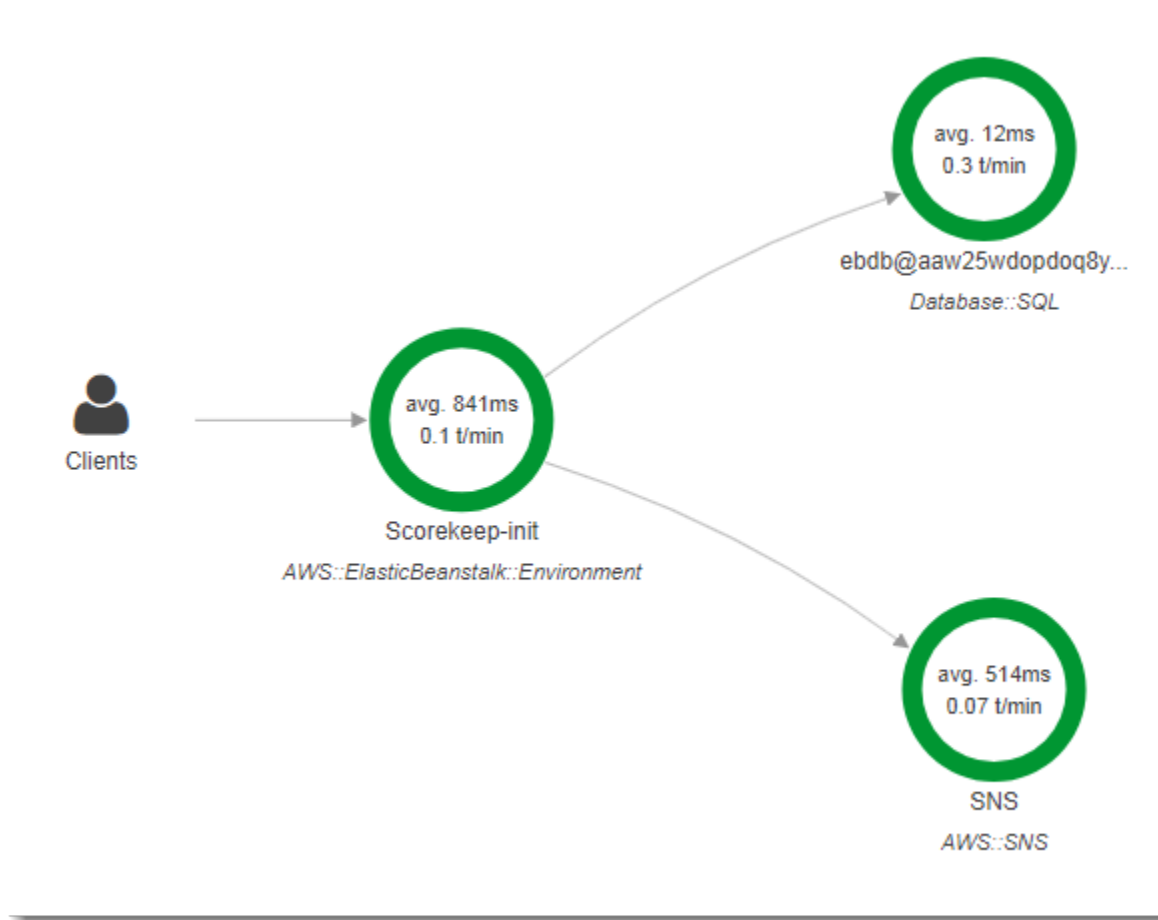
    # Receive message from SQS queue
    response = sqs.receive_message(
        QueueUrl=queue_url,
        AttributeNames=[
            'SentTimestamp'
        ],
        MaxNumberOfMessages=1,
        MessageAttributeNames=[
            'All'
        ],
        VisibilityTimeout=0,
        WaitTimeSeconds=0
    )
    ...
```

Instrumentation du code de démarrage

Le SDK X-Ray pour Java crée automatiquement des segments pour les demandes entrantes. En ce qui concerne les demandes, vous pouvez utiliser des clients instrumentés et enregistrer les sous-

segments sans problème. Cependant, si vous essayez d'utiliser un client instrumenté dans le code de démarrage, vous obtiendrez un [SegmentNotFoundException](#).

Le code de démarrage s'exécute en dehors du flux de demande/réponse standard d'une application web. Vous devez donc créer des segments manuellement pour l'instrumenter. Scorekeep présente l'instrumentation du code de démarrage dans ses fichier WebConfig. Scorekeep appelle une base de données SQL et Amazon SNS au démarrage.



La WebConfig classe par défaut crée un abonnement Amazon SNS pour les notifications. Pour fournir un segment dans lequel le SDK X-Ray peut écrire lorsque le client Amazon SNS est utilisé, Scorekeep beginSegment appelle endSegment et sur l'enregistreur global.

Exemple [src/main/java/scorekeep/WebConfig.java](#)— Client AWS SDK instrumenté dans le code de démarrage

```
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();

```

Dans `RdsWebConfig`, que Scorekeep utilise lorsqu'une base de données Amazon RDS est connectée, la configuration crée également un segment pour le client SQL qu'Hibernate utilise lorsqu'il applique le schéma de base de données au démarrage.

Exemple [src/main/java/scorekeep/RdsWebConfig.java](#)— Client de base de données SQL instrumenté dans le code de démarrage

```

@PostConstruct
public void schemaExport() {
    EntityManagerFactoryImpl entityManagerFactoryImpl = (EntityManagerFactoryImpl)
localContainerEntityManagerFactoryBean.getNativeEntityManagerFactory();
    SessionFactoryImplementor sessionFactoryImplementor =
entityManagerFactoryImpl.getSessionFactory();
    StandardServiceRegistry standardServiceRegistry =
sessionFactoryImplementor.getSessionFactoryOptions().getServiceRegistry();
    MetadataSources metadataSources = new MetadataSources(new
BootstrapServiceRegistryBuilder().build());
    metadataSources.addAnnotatedClass(GameHistory.class);
    MetadataImplementor metadataImplementor = (MetadataImplementor)
metadataSources.buildMetadata(standardServiceRegistry);
    SchemaExport schemaExport = new SchemaExport(standardServiceRegistry,
metadataImplementor);

    AWSXRay.beginSegment("Scorekeep-init");
    schemaExport.create(true, true);
    AWSXRay.endSegment();
}

```

`SchemaExport` s'exécute automatiquement et utilise un client SQL. Dans la mesure où le client est instrumenté, Scorekeep doit remplacer la mise en œuvre par défaut et fournir un segment qui est utilisé par le kit de développement logiciel lorsque le client est appelé.

Instrumentation de scripts

Vous pouvez également instrumenter le code qui ne fait pas partie de l'application. Lorsque le daemon X-Ray est en cours d'exécution, il transmet tous les segments qu'il reçoit à X-Ray, même s'ils ne sont pas générés par le SDK de X-Ray. Scorekeep utilise ses propres scripts pour instrumenter la build qui compile l'application pendant le déploiement.

Exemple [bin/build.sh](#)— Script de construction instrumenté

```
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](#) et [xray_success.py](#) sont de simples scripts Python qui construisent des objets de segment, les convertissent en documents JSON et les envoient au démon via UDP. Si la compilation de Gradle échoue, vous pouvez trouver le message d'erreur en cliquant sur le nœud scorekeep-build dans la carte de suivi de la console X-Ray.



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

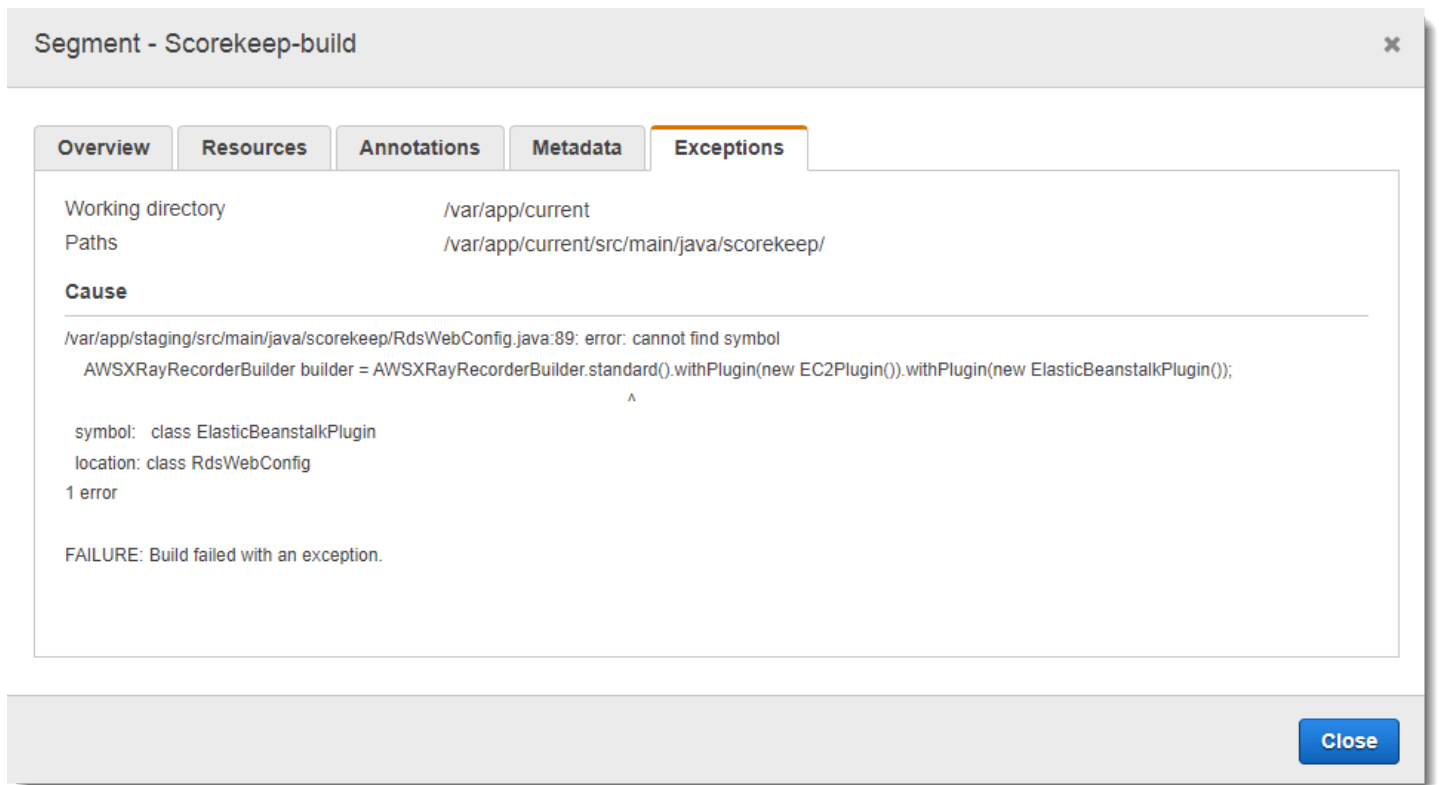
▼ Scorekeep-build

Scorekeep-build

-

14.6 sec

--



Instrumentation d'un client d'application web

Dans la [xray-cognito](#) succursale, Scorekeep utilise Amazon Cognito pour permettre aux utilisateurs de créer un compte et de s'y connecter afin de récupérer leurs informations utilisateur depuis un groupe d'utilisateurs Amazon Cognito. Lorsqu'un utilisateur se connecte, Scorekeep utilise un pool d'identités Amazon Cognito pour obtenir des informations d'identification AWS temporaires à utiliser avec le AWS SDK pour JavaScript

Le pool d'identités est configuré pour que les utilisateurs connectés puissent écrire des données de suivi sur AWS X-Ray. L'application web utilise ces informations d'identification pour enregistrer l'ID de l'utilisateur connecté, le chemin du navigateur et la vue des appels du client vers l'API Scorekeep.

La plupart des opérations sont effectuées dans une classe de service nommée `xray`. Cette classe de service fournit des méthodes permettant de générer les identifiants requis, de créer des segments en cours, de finaliser des segments et d'envoyer des documents de segment à l'API X-Ray.

Exemple [public/xray.js](#)— Enregistrez et téléchargez des segments

```
...
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);
    }
  })
}

```

$$\left. \begin{array}{l} \} \\ \} \end{array} \right) \}$$

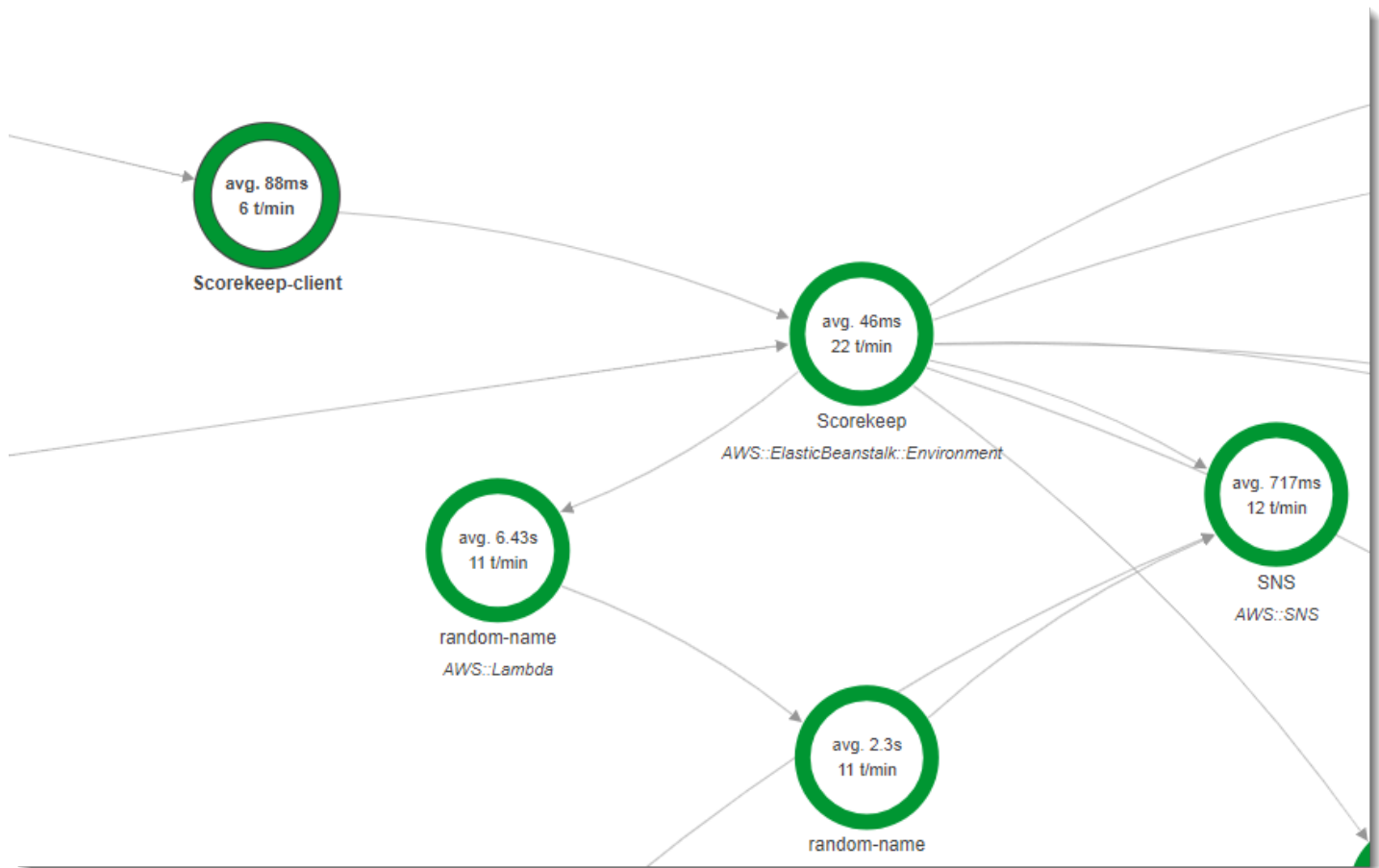
Ces méthodes sont appelées dans les fonctions d'en-tête et `transformResponse` dans les services de ressource utilisés par l'application web pour appeler l'API Scorekeep. Pour inclure le segment client dans la même trace que le segment généré par l'API, l'application Web doit inclure l'ID de trace et l'ID de segment dans un [en-tête de suivi](#) (`X-Amzn-Trace-Id`) que le SDK X-Ray peut lire. Lorsque l'application Java instrumentée reçoit une demande avec cet en-tête, le SDK X-Ray pour Java utilise le même identifiant de trace et fait du segment du client de l'application Web le parent de son segment.

Exemple [public/app/services.js](#)— Enregistrement de segments pour les appels de ressources angulaires et écriture d'en-têtes de suivi

```
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);
    },
  },
});
...

```

La carte de trace obtenue inclut un nœud pour le client de l'application Web.



Les suivis qui contiennent des segments provenant de l'application web affichent l'URL que l'utilisateur voit dans le navigateur (chemins commençant par `/#/`). Sans l'instrumentation du client, vous obtenez uniquement l'URL de la ressource d'API que l'application web appelle (chemins commençant par `/api/`).

Trace overview

Group by: URL

URL	Avg response time
http://scorekeep.elasticbeanstalk.com/#/	86.2 ms
http://scorekeep.elasticbeanstalk.com/#/session/4ORP7OB5/47H4SETD	58.5 ms
http://scorekeep.elasticbeanstalk.com/#/game/4ORP7OB5/A94SAFFD/47H4SETD	255 ms

Utilisation de clients instrumentés dans les threads de travail

Scorekeep utilise un fil de discussion pour publier une notification sur Amazon SNS lorsqu'un utilisateur gagne une partie. La publication de la notification prend plus de temps que le reste des opérations de demande combinées et n'affecte pas le client ou l'utilisateur. Par conséquent, l'exécution de la tâche de manière asynchrone est un bon moyen d'améliorer le temps de réponse.

Cependant, le SDK X-Ray pour Java ne sait pas quel segment était actif lors de la création du thread. Par conséquent, lorsque vous essayez d'utiliser le AWS SDK pour Java client instrumenté dans le thread, cela lance un `SegmentNotFoundException` et fait planter le thread.

Exemple 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:
...

```

Pour résoudre ce problème, l'application `GetTraceEntity` obtient une référence au segment dans le thread principal et `Entity.run()` exécute en toute sécurité le code du thread de travail avec accès au contexte du segment.

Exemple [src/main/java/scorekeep/MoveFactory.java](#)— Transmission du contexte de trace à un thread de travail

```
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);
        });
    }
};

```

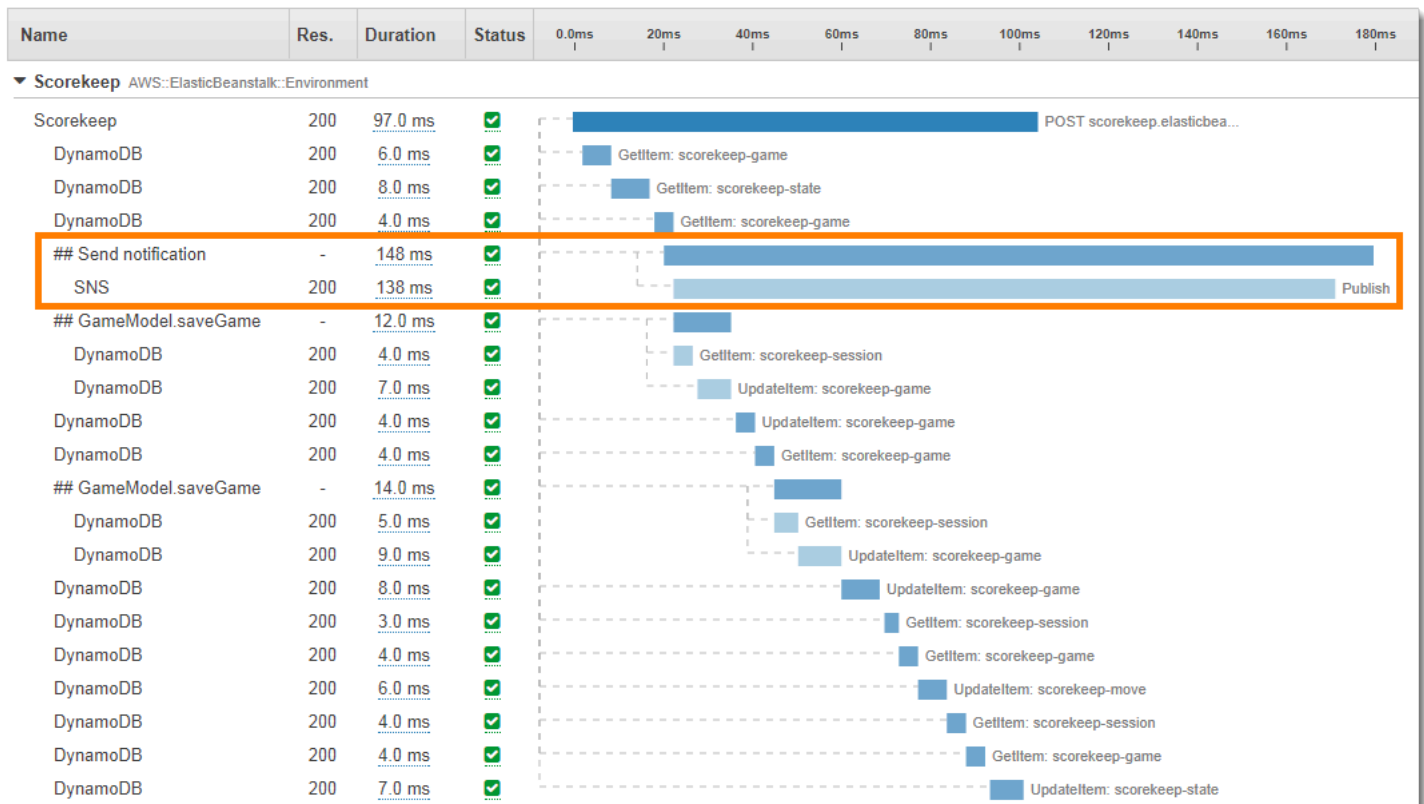


```

    AWSXRay.endSubsegment();
  }
}

```

La demande étant désormais résolue avant l'appel à Amazon SNS, l'application crée un sous-segment distinct pour le thread. Cela empêche le SDK X-Ray de fermer le segment avant d'enregistrer la réponse d'Amazon SNS. Si aucun sous-segment n'est ouvert lorsque Scorekeep a résolu la demande, la réponse d'Amazon SNS risque d'être perdue.



Pour plus d'informations sur le multithreading, consultez [Transmission de contexte de segment entre threads dans une application multithreads](#).

AWS X-Ray démon

Note

Vous pouvez désormais utiliser l' CloudWatch agent pour collecter des métriques, des journaux et des traces à partir d' EC2 instances Amazon et de serveurs sur site. CloudWatch les versions 1.300025.0 et ultérieures de l'agent peuvent collecter des traces depuis notre client [OpenTelemetryX-Ray](#) et SDKs les envoyer à X-Ray. L'utilisation de l' CloudWatch agent au lieu du daemon AWS Distro for OpenTelemetry (ADOT) Collector ou X-Ray pour collecter des traces peut vous aider à réduire le nombre d'agents que vous gérez. Consultez la rubrique [CloudWatch relative aux agents](#) dans le guide de CloudWatch l'utilisateur pour plus d'informations.

Le AWS X-Ray daemon est une application logicielle qui écoute le trafic sur le port UDP 2000, collecte les données de segment brutes et les transmet à l'API. AWS X-Ray Le daemon fonctionne conjointement avec le AWS X-Ray SDKs et doit être en cours d'exécution pour que les données envoyées par le SDKs puissent atteindre le service X-Ray. Le daemon X-Ray est un projet open source. Vous pouvez suivre le projet et soumettre des problèmes et des pull requests sur GitHub : github.com/aws/aws-xray-daemon

AWS Lambda Activez et AWS Elastic Beanstalk utilisez l'intégration de ces services à X-Ray pour exécuter le daemon. Lambda exécute le démon automatiquement chaque fois qu'une fonction est invoquée pour une requête échantillonnée. Sur Elastic [Beanstalk, XRayEnabled utilisez l'option de configuration pour exécuter le](#) daemon sur les instances de votre environnement. Pour plus d'informations, veuillez consulter la rubrique

Pour exécuter le daemon X-Ray localement, sur site ou sur un autre support Services AWS, téléchargez-le, [exécutez-le](#), puis [autorisez-le](#) à télécharger des documents segmentés sur X-Ray.

Téléchargement du démon

Vous pouvez télécharger le daemon depuis Amazon S3, Amazon ECR ou Docker Hub, puis l'exécuter localement ou l'installer sur une EC2 instance Amazon au lancement.

Amazon S3

Installeurs et exécutables du daemon X-Ray

- Linux (exécutable) — [aws-xray-daemon-linux-3.x.zip\(sig\)](#)
- Linux (programme d'installation RPM) — [aws-xray-daemon-3.x.rpm](#)
- Linux (installateur DEB) — [aws-xray-daemon-3.x.deb](#)
- Linux (ARM64, exécutable) — [aws-xray-daemon-linux-arm64-3.x.zip\(sig\)](#)
- Linux (ARM64, installateur RPM) — [aws-xray-daemon-arm64-3.x.rpm](#)
- Linux (ARM64installateur DEB) — [aws-xray-daemon-arm64-3.x.deb](#)
- OS X (exécutable) — [aws-xray-daemon-macos-3.x.zip\(signature\)](#)
- Windows (exécutable) — [aws-xray-daemon-windows-process-3.x.zip\(sig\)](#)
- Windows (service) — [aws-xray-daemon-windows-service-3.x.zip\(sig\)](#)

Ces liens pointent toujours vers la dernière version 3.x du daemon. Pour télécharger une version spécifique, procédez comme suit :

- Si vous souhaitez télécharger une version antérieure 3.3.0, remplacez-la 3.x par le numéro de version. Par exemple, 2.1.0. Avant la version 3.3.0, la seule architecture disponible était arm64. Par exemple : 2.1.0 et arm64.
- Si vous souhaitez télécharger une version après l'autre 3.3.0, remplacez-la 3.x par le numéro de version et arch par le type d'architecture.

Les ressources X-Ray sont répliquées dans des buckets dans chaque région prise en charge. Pour utiliser le compartiment le plus proche de vous ou de vos AWS ressources, remplacez la région dans les liens ci-dessus par votre région.

```
https://s3.us-west-2.amazonaws.com/aws-xray-assets.us-west-2/xray-daemon/aws-xray-daemon-3.x.rpm
```

Amazon ECR

Depuis la version 3.2.0, le daemon est disponible sur [Amazon](#) ECR. Avant de récupérer une image, vous devez [authentifier votre client docker](#) auprès du registre public Amazon ECR.

Extrayez la dernière balise de version 3.x publiée en exécutant la commande suivante :

```
docker pull public.ecr.aws/xray/aws-xray-daemon:3.x
```

Les versions antérieures ou alpha peuvent être téléchargées en les remplaçant par un numéro de version spécifique alpha ou en le 3.x remplaçant par un numéro de version spécifique. Il n'est pas recommandé d'utiliser une image de démon avec une balise alpha dans un environnement de production.

Docker Hub

Le daemon se trouve sur [Docker Hub](#). Pour télécharger la dernière version 3.x publiée, exécutez la commande suivante :

```
docker pull amazon/aws-xray-daemon:3.x
```

Les versions précédentes du daemon peuvent être publiées en les 3.x remplaçant par la version souhaitée.

Vérification de la signature de l'archive du démon

Les fichiers signature GPG sont inclus pour les ressources compressées du démon dans des archives ZIP. La clé publique se trouve ici : [aws-xray.gpg](#).

Vous pouvez utiliser la clé publique pour vérifier que l'archive ZIP du démon est l'archive originale non modifiée. Tout d'abord, importez la clé publique avec [GnuPG](#).

Pour importer la clé publique

1. Téléchargez la clé publique.

```
$ BUCKETURL=https://s3.us-east-2.amazonaws.com/aws-xray-assets.us-east-2
$ wget $BUCKETURL/xray-daemon/aws-xray.gpg
```

2. Importez la clé publique dans votre porte-clés.

```
$ 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
```

Utilisez la clé importée pour vérifier la signature de l'archive ZIP du démon.

Pour vérifier une signature d'archive

1. Téléchargez l'archive et le fichier signature.

```
$ 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. Exécutez `gpg --verify` pour vérifier la signature.

```
$ 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
```

Notez l'avertissement sur la confiance. Une clé est uniquement approuvée si vous ou une personne de confiance l'a signée. Cela ne signifie pas que la signature n'est pas valide, mais seulement que vous n'avez pas vérifié la clé publique.

Exécution du démon

Exécutez le démon en local via la ligne de commande. Utilisez l'option `-o` pour l'exécuter en mode local et `-n` pour définir la région.

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

Pour obtenir des instructions détaillées spécifiques à la plateforme, consultez les rubriques suivantes :

- Linux (local) — [Exécution du daemon X-Ray sous Linux](#)
- Windows (local) — [Exécution du daemon X-Ray sous Windows](#)
- Elastic Beanstalk — [Exécution du daemon X-Ray sur AWS Elastic Beanstalk](#)
- Amazon EC2 — [Exécution du daemon X-Ray sur Amazon EC2](#)
- Amazon ECS — [Exécution du daemon X-Ray sur Amazon ECS](#)

Vous pouvez personnaliser encore plus le comportement du démon à l'aide des options de ligne de commande ou d'un fichier de configuration. Consultez [Configuration du AWS X-Ray daemon](#) pour plus de détails.

Autoriser le daemon à envoyer des données à X-Ray

Le daemon X-Ray utilise le AWS SDK pour télécharger les données de suivi vers X-Ray, et il a besoin AWS d'informations d'identification avec autorisation pour le faire.

Sur Amazon EC2, le daemon utilise automatiquement le rôle de profil d'instance de l'instance. Pour plus d'informations sur les informations d'identification requises pour exécuter le démon localement, consultez la section [Exécution locale de votre application](#).

Si vous spécifiez des informations d'identification dans plusieurs emplacements (fichier d'informations d'identification, profil d'instance ou variables d'environnement), la chaîne de fournisseur du kit SDK détermine les informations d'identification qui sont utilisées. Pour plus d'informations sur la fourniture d'informations d'identification au SDK, consultez [Spécifier les informations d'identification](#) dans le guide du développeur du AWS SDK for Go.

Le rôle IAM ou l'utilisateur auquel appartiennent les informations d'identification du démon doit disposer de l'autorisation d'écrire des données sur le service en votre nom.

- Pour utiliser le daemon sur Amazon EC2, créez un nouveau rôle de profil d'instance ou ajoutez la politique gérée à un rôle existant.
- Pour utiliser le daemon sur Elastic Beanstalk, ajoutez la politique gérée au rôle de profil d'instance par défaut d'Elastic Beanstalk.
- Pour exécuter le daemon localement, consultez la section [Exécution locale de votre application](#).

Pour de plus amples informations, veuillez consulter [Gestion des identités et des accès pour AWS X-Ray](#).

Journaux du démon X-Ray

Le daemon produit des informations sur sa configuration actuelle et les segments auxquels il envoie des informations. 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)
```

Par défaut, le démon génère des journaux dans STDOUT. Si vous exécutez le démon en arrière-plan, utilisez l'option de ligne de commande `--log-file` ou un fichier de configuration pour définir le chemin d'accès au fichier journal. Vous pouvez également définir le niveau de journalisation et désactiver la rotation des journaux. Pour obtenir des instructions, consultez [Configuration du AWS X-Ray daemon](#).

Sur Elastic Beanstalk, la plateforme définit l'emplacement des journaux des démons. Consultez [Exécution du daemon X-Ray sur AWS Elastic Beanstalk](#) pour plus de détails.

Configuration du AWS X-Ray daemon

Vous pouvez utiliser les options de ligne de commande ou un fichier de configuration pour personnaliser le comportement du daemon X-Ray. La plupart des options sont disponibles en utilisant les deux méthodes, mais certaines ne sont disponibles que dans les fichiers de configuration et d'autres uniquement sur la ligne de commande.

Pour commencer, la seule option que vous devez connaître est `-n` ou `--region`, que vous utilisez pour définir la région utilisée par le daemon pour envoyer les données de suivi à X-Ray.

```
~/xray-daemon$ ./xray -n us-east-2
```

Si vous exécutez le démon localement, c'est-à-dire pas sur Amazon EC2, vous pouvez ajouter l'option permettant de ne pas vérifier les informations d'identification du profil de l'instance afin que le démon soit prêt plus rapidement.

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

Le reste des options de ligne de commande vous permettent de configurer la journalisation, d'écouter sur un autre port, de limiter la quantité de mémoire utilisée par le démon ou d'assumer un rôle pour envoyer des données de suivi à un autre compte.

Vous pouvez transmettre un fichier de configuration au daemon pour accéder aux options de configuration avancées et notamment limiter le nombre d'appels simultanés à X-Ray, désactiver la rotation des journaux et envoyer le trafic vers un proxy.

Sections

- [Variables d'environnement prises en charge](#)
- [Utilisation d'options de ligne de commande](#)
- [Utilisation d'un fichier de configuration](#)

Variables d'environnement prises en charge

Le daemon X-Ray prend en charge les variables d'environnement suivantes :

- `AWS_REGION`— Spécifie le point [Région AWS](#) de terminaison du service X-Ray.
- `HTTPS_PROXY`— Spécifie une adresse proxy par laquelle le démon doit télécharger des segments. Il peut s'agir des noms de domaine DNS ou des adresses IP et des numéros de port utilisés par vos serveurs proxy.

Utilisation d'options de ligne de commande

Transmettez ces options au démon lorsque vous l'exécutez en local ou avec un script de données utilisateur.

Options de ligne de commande

- `-b, --bind` — Écoutez les documents segmentés sur un autre port UDP.

```
--bind "127.0.0.1:3000"
```

Par défaut —2000.

- `-t, --bind-tcp` — Écoutez les appels vers le service X-Ray sur un autre port TCP.

```
-bind-tcp "127.0.0.1:3000"
```

Par défaut —2000.

- `-c, --config` — Charge un fichier de configuration depuis le chemin spécifié.

```
--config "/home/ec2-user/xray-daemon.yaml"
```

- `-f, --log-file` — Affiche les journaux dans le chemin de fichier spécifié.


```
--log-file "/var/log/xray-daemon.log"
```

- `-l, --log-level` — Niveau de journalisation, du plus détaillé au moins détaillé : dev, debug, info, warn, error, prod.

```
--log-level warn
```

Par défaut — prod

- `-m, --buffer-memory` — Modifiez la quantité de mémoire en Mo que les tampons peuvent utiliser (minimum 3).

```
--buffer-memory 50
```

Par défaut : 1 % de la mémoire disponible.

- `-o, --local-mode` — Ne vérifiez pas les métadonnées des EC2 instances.
- `-r, --role-arn` — Assumez le rôle IAM spécifié pour télécharger des segments vers un autre compte.

```
--role-arn "arn:aws:iam::123456789012:role/xray-cross-account"
```

- `-a, --resource-arn` — Amazon Resource Name (ARN) de la AWS ressource exécutant le daemon.
- `-p, --proxy-address` — Téléchargez des segments AWS X-Ray via un proxy. Le protocole du serveur proxy doit être spécifié.

```
--proxy-address "http://192.0.2.0:3000"
```

- `-n, --region` — Envoyez des segments au service X-Ray d'une région spécifique.
- `-v, --version` — Affiche la version du AWS X-Ray démon.
- `-h, --help` — Affiche l'écran d'aide.

Utilisation d'un fichier de configuration

Vous pouvez également utiliser un fichier au format YAML pour configurer le démon. Transmettez le fichier de configuration au démon à l'aide de l'option `-c`.

```
~$ ./xray -c ~/xray-daemon.yaml
```

Configuration des options du fichier

- **TotalBufferSizeMB**— Taille maximale de la mémoire tampon en Mo (minimum 3). Choisissez 0 pour utiliser 1 % de la mémoire hôte.
- **Concurrency**— Nombre maximum d'appels simultanés AWS X-Ray pour télécharger des documents segmentés.
- **Region**— Mettez des segments en AWS X-Ray service dans une région spécifique.
- **Socket**— Configure la liaison du daemon.
 - **UDPEndress**— Changez le port sur lequel le daemon écoute.
 - **TCPAddress**— Écoutez les [appels vers le service X-Ray](#) sur un autre port TCP.
- **Logging**— Configurez le comportement de journalisation.
 - **LogRotation**— Réglé sur `false` pour désactiver la rotation des journaux.
 - **LogLevel**— Modifiez le niveau de journalisation, du plus détaillé au moins détaillé : `dev`, `info`, `debug`, `prod`, `warn`, `error`. `prod` La valeur par défaut est `prod`, ce qui équivaut à `info`.
 - **LogPath**— Affiche les journaux dans le chemin de fichier spécifié.
- **LocalMode**— Réglé sur `true` pour ignorer la vérification des métadonnées de l' EC2 instance.
- **ResourceARN**— Amazon Resource Name (ARN) de la AWS ressource exécutant le daemon.
- **RoleARN**— Assumez le rôle IAM spécifié pour télécharger des segments vers un autre compte.
- **ProxyAddress**— Téléchargez des segments AWS X-Ray via un proxy.
- **Endpoint**— Modifiez le point de terminaison du service X-Ray auquel le démon envoie les documents de segment.
- **NoVerifySSL**— Désactive la vérification des certificats TLS.
- **Version**— Version du format du fichier de configuration du démon. La version du format de fichier est un champ obligatoire.

Exemple xray-daemon.yaml

Ce fichier de configuration remplace le port d'écoute du démon par 3000, désactive les vérifications des métadonnées d'instance, définit un rôle à utiliser pour le chargement des segments et modifie les options de région et de journalisation.

```
Socket :
```

```
UDPEndPoint: "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
```

Exécution du daemon X-Ray en local

Vous pouvez exécuter le AWS X-Ray démon localement sous Linux, macOS, Windows ou dans un conteneur Docker. Exécutez le daemon pour transmettre les données de suivi à X-Ray lorsque vous développez et testez votre application instrumentée. Téléchargez et extrayez le démon en utilisant les instructions indiquées [ici](#).

Lorsqu'il est exécuté localement, le démon peut lire les informations d'identification à partir d'un fichier d'informations d'identification du AWS SDK (.aws/credentials dans votre répertoire utilisateur) ou à partir de variables d'environnement. Pour de plus amples informations, veuillez consulter [Autoriser le daemon à envoyer des données à X-Ray](#).

Le démon écoute les données UDP sur le port 2000. Vous pouvez modifier le port et les autres options à l'aide d'un fichier de configuration et des options de ligne de commande. Pour de plus amples informations, veuillez consulter [Configuration du AWS X-Ray daemon](#).

Exécution du daemon X-Ray sous Linux

Vous pouvez exécuter l'exécutable du démon à partir de la ligne de commande. Utilisez l'option `-o` pour l'exécuter en mode local et `-n` pour définir la région.

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

Pour exécuter le démon en arrière-plan, utilisez `&`.

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

Terminez un processus démon en cours d'exécution en arrière-plan avec `kill`.

```
~$ pkill xray
```

Exécution du daemon X-Ray dans un conteneur Docker

Pour exécuter le démon en local dans un conteneur Docker, enregistrez le texte suivant dans un fichier nommé `Dockerfile`. Téléchargez l'[image d'exemple](#) complète sur Amazon ECR. Consultez la section [Téléchargement du daemon](#) pour plus d'informations.

Exemple Dockerfile — Amazon Linux

```
FROM amazonlinux
RUN yum install -y unzip
RUN curl -o daemon.zip https://s3.us-east-2.amazonaws.com/aws-xray-assets.us-east-2/
xray-daemon/aws-xray-daemon-linux-3.x.zip
RUN unzip daemon.zip && cp xray /usr/bin/xray
ENTRYPOINT ["/usr/bin/xray", "-t", "0.0.0.0:2000", "-b", "0.0.0.0:2000"]
EXPOSE 2000/udp
EXPOSE 2000/tcp
```

Générez l'image de conteneur avec `docker build`.

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

Exécutez l'image dans un conteneur avec `docker run`.

```
~/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
```

Cette commande utilise les options suivantes :

- `--attach STDOUT`— Affiche le résultat du démon dans le terminal.
- `-v ~/.aws/:/root/.aws/:ro`— Donnez au conteneur un accès en lecture seule au `.aws` répertoire pour lui permettre de lire vos informations d'identification du AWS SDK.

- `AWS_REGION=us-east-2`— Définissez la variable d'`AWS_REGION` d'environnement pour indiquer au démon la région à utiliser.
- `--net=host`— Connectez le conteneur au `host` réseau. Les conteneurs sur le réseau hôte peuvent communiquer entre eux sans publication des ports.
- `-p 2000:2000/udp`— Mappez le port UDP 2000 de votre machine au même port du conteneur. Ceci n'est pas requis pour que les conteneurs figurant sur le même réseau communiquent, mais cela vous permet d'envoyer des segments au démon [à partir de la ligne de commande](#) ou à partir d'une application qui ne s'exécute pas dans Docker.
- `--name xray-daemon`— Nommez le conteneur `xray-daemon` au lieu de générer un nom aléatoire.
- `-o`(après le nom de l'image) — Ajoutez l'`-o`option au point d'entrée qui exécute le démon dans le conteneur. Cette option indique au démon de s'exécuter en mode local pour l'empêcher d'essayer de lire les métadonnées des EC2 instances Amazon.

Pour arrêter le démon, utilisez `docker stop`. Si vous apportez des modifications au fichier `Dockerfile` et que vous générez une nouvelle image, vous devez supprimer le conteneur existant pour pouvoir en créer un autre avec le même nom. Utilisez `docker rm` pour supprimer le conteneur.

```
$ docker stop xray-daemon
$ docker rm xray-daemon
```

Exécution du daemon X-Ray sous Windows

Vous pouvez exécuter l'exécutable du démon à partir de la ligne de commande. Utilisez l'option `-o` pour l'exécuter en mode local et `-n` pour définir la région.

```
> .\xray_windows.exe -o -n us-east-2
```

Utilisez un PowerShell script pour créer et exécuter un service pour le démon.

Exemple PowerShell script - Windows

```
if ( Get-Service "AWSXRayDaemon" -ErrorAction SilentlyContinue ){
    sc.exe stop AWSXRayDaemon
    sc.exe delete AWSXRayDaemon
}
```

```
if ( Get-Item -path aws-xray-daemon -ErrorAction SilentlyContinue ) {  
    Remove-Item -Recurse -Force aws-xray-daemon  
}  
  
$currentLocation = Get-Location  
$zipFileName = "aws-xray-daemon-windows-service-3.x.zip"  
$zipPath = "$currentLocation\$zipFileName"  
$destPath = "$currentLocation\aws-xray-daemon"  
$daemonPath = "$destPath\xray.exe"  
$daemonLogPath = "C:\inetpub\wwwroot\xray-daemon.log"  
$url = "https://s3.dualstack.us-west-2.amazonaws.com/aws-xray-assets.us-west-2/xray-  
daemon/aws-xray-daemon-windows-service-3.x.zip"  
  
Invoke-WebRequest -Uri $url -OutFile $zipPath  
Add-Type -Assembly "System.IO.Compression.FileSystem"  
[io.compression.zipfile]::ExtractToDirectory($zipPath, $destPath)  
  
sc.exe create AWSXRayDaemon binPath= "$daemonPath -f $daemonLogPath"  
sc.exe start AWSXRayDaemon
```

Exécution du daemon X-Ray sous OS X

Vous pouvez exécuter l'exécutable du démon à partir de la ligne de commande. Utilisez l'option `-o` pour l'exécuter en mode local et `-n` pour définir la région.

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

Pour exécuter le démon en arrière-plan, utilisez `&`.

```
~/xray-daemon$ ./xray_mac -o -n us-east-2 &
```

Utilisez `nohup` pour empêcher la mise hors service du démon lorsque le terminal est fermé.

```
~/xray-daemon$ nohup ./xray_mac &
```

Exécution du daemon X-Ray sur AWS Elastic Beanstalk

Pour relayer les données de suivi de votre application vers AWS X-Ray, vous pouvez exécuter le daemon X-Ray sur les instances Amazon de votre environnement Elastic Beanstalk. EC2 Pour

obtenir la liste des plateformes prises en charge, consultez [la section Configuration du AWS X-Ray débogage](#) dans le guide du AWS Elastic Beanstalk développeur.

 Note

Le démon utilise le profil d'instance de votre environnement pour les autorisations. Pour obtenir des instructions sur l'ajout d'autorisations au profil d'instance Elastic Beanstalk, consultez. [Autoriser le daemon à envoyer des données à X-Ray](#)

Les plateformes Elastic Beanstalk fournissent une option de configuration que vous pouvez configurer pour exécuter le daemon automatiquement. Vous pouvez activer le daemon dans un fichier de configuration de votre code source ou en choisissant une option dans la console Elastic Beanstalk. Lorsque vous activez l'option de configuration, le démon est installé sur l'instance et s'exécute en tant que service.

La version incluse sur les plateformes Elastic Beanstalk n'est peut-être pas la dernière version. Consultez la [rubrique Plateformes prises en charge](#) pour identifier la version du démon disponible pour votre configuration de plateforme.

Elastic Beanstalk ne fournit pas le daemon X-Ray sur la plateforme Multicontainer Docker (Amazon ECS).

Utilisation de l'intégration Elastic Beanstalk X-Ray pour exécuter le daemon X-Ray

Utilisez la console pour activer l'intégration de X-Ray ou configurez-la dans le code source de votre application à l'aide d'un fichier de configuration.

Pour activer le daemon X-Ray dans la console Elastic Beanstalk

1. Ouvrez la [console Elastic Beanstalk](#).
2. Accédez à la [console de gestion](#) de votre environnement.
3. Choisissez Configuration.
4. Choisissez Software Settings.
5. Pour Daemon X-Ray, choisissez Enabled.
6. Choisissez Appliquer.

Vous pouvez inclure un fichier de configuration dans votre code source pour que votre configuration soit portable entre les environnements.

Exemple `.ebextensions/xray-daemon.config`

```
option_settings:
  aws:elasticbeanstalk:xray:
    XRayEnabled: true
```

Elastic Beanstalk transmet un fichier de configuration au daemon et génère des journaux vers un emplacement standard.

Sur les plateformes Windows Server

- Fichier de configuration — `C:\Program Files\Amazon\XRay\cfg.yaml`
- Journaux — `c:\Program Files\Amazon\XRay\logs\xray-service.log`

Sur les plateformes Linux

- Fichier de configuration — `/etc/amazon/xray/cfg.yaml`
- Journaux — `/var/log/xray/xray.log`

Elastic Beanstalk fournit des outils permettant d'extraire les journaux AWS Management Console d'instance depuis la ligne de commande `or`. Vous pouvez demander à Elastic Beanstalk d'inclure les journaux du daemon X-Ray en ajoutant une tâche avec un fichier de configuration.

Exemple `.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
```

Exemple `.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

```

Pour plus d'informations, [consultez la section Visualisation des journaux des AWS Elastic Beanstalk instances Amazon de votre environnement Elastic EC2 Beanstalk](#) dans le manuel du développeur.

Téléchargement et exécution manuels du daemon X-Ray (niveau avancé)

Si le daemon X-Ray n'est pas disponible pour la configuration de votre plateforme, vous pouvez le télécharger depuis Amazon S3 et l'exécuter avec un fichier de configuration.

Utilisez un fichier de configuration Elastic Beanstalk pour télécharger et exécuter le daemon.

Exemple .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
```

Ces exemples ajoutent également le fichier journal du démon à la tâche Elastic Beanstalk tail logs, afin qu'il soit inclus lorsque vous demandez des journaux via la console ou l'interface de ligne de commande Elastic Beanstalk (EB CLI).

Exécution du daemon X-Ray sur Amazon EC2

Vous pouvez exécuter le daemon X-Ray sur les systèmes d'exploitation suivants sur Amazon EC2 :

- Amazon Linux
- Ubuntu
- Windows Server (2012 R2 et versions ultérieures)

Utilisez un profil d'instance pour autoriser le démon à télécharger des données de trace vers X-Ray. Pour de plus amples informations, veuillez consulter [Autoriser le daemon à envoyer des données à X-Ray](#).

Utilisez un script des données utilisateur pour exécuter automatiquement le démon lorsque vous lancez l'instance.

Exemple Script des données utilisateur - 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
```

Exemple Script des données utilisateur - 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>

```

Exécution du daemon X-Ray sur Amazon ECS

Dans Amazon ECS, créez une image Docker qui exécute le démon X-Ray, téléchargez-la dans un référentiel d'images Docker, puis déployez-la sur votre cluster Amazon ECS. Vous pouvez utiliser des mappages de ports et des paramètres de mode réseau dans votre fichier de définition de tâche pour autoriser votre application à communiquer avec le conteneur du démon.

Utilisation de l'image Docker officielle de l'

X-Ray fournit une [image de conteneur](#) Docker sur Amazon ECR que vous pouvez déployer parallèlement à votre application. Consultez la section [Téléchargement du daemon](#) pour plus d'informations.

Exemple Définition de tâche

```

{
  "name": "xray-daemon",
  "image": "amazon/aws-xray-daemon",

```

```
"cpu": 32,  
"memoryReservation": 256,  
"portMappings" : [  
  {  
    "hostPort": 0,  
    "containerPort": 2000,  
    "protocol": "udp"  
  }  
]  
}
```

Création et génération d'une image Docker

Pour une configuration personnalisée, vous devrez peut-être définir votre propre image Docker.

Ajoutez des politiques gérées à votre rôle de tâche pour autoriser le démon à télécharger des données de suivi vers X-Ray. Pour de plus amples informations, veuillez consulter [Autoriser le daemon à envoyer des données à X-Ray](#).

Utilisez l'un des fichiers Dockerfile suivants pour créer une image qui exécute le démon.

Exemple Dockerfile — Amazon Linux

```
FROM amazonlinux  
RUN yum install -y unzip  
RUN curl -o daemon.zip https://s3.us-east-2.amazonaws.com/aws-xray-assets.us-east-2/  
xray-daemon/aws-xray-daemon-linux-3.x.zip  
RUN unzip daemon.zip && cp xray /usr/bin/xray  
ENTRYPOINT ["/usr/bin/xray", "-t", "0.0.0.0:2000", "-b", "0.0.0.0:2000"]  
EXPOSE 2000/udp  
EXPOSE 2000/tcp
```

Note

Les indicateurs `-t` et `-b` sont requis pour spécifier une adresse de liaison afin d'écouter le bouclage d'un environnement à plusieurs conteneurs.

Exemple Dockerfile — Ubuntu

Pour les dérivés de Debian, vous devez également installer les certificats de l'autorité de certification pour éviter toute erreur lors du téléchargement du programme d'installation.

```
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
```

Dans votre définition de tâche, la configuration dépend du mode de mise en réseau que vous utilisez. La mise en réseau par pont est le mode par défaut et peut être utilisé dans votre VPC par défaut. Dans un réseau en pont, définissez la variable d'AWS_XRAY_DAEMON_ADDRESS pour indiquer au SDK X-Ray le port conteneur à référencer et définissez le port hôte. Par exemple, vous pouvez publier le port UDP 2000 et créer un lien depuis votre conteneur d'applications vers le conteneur du démon.

Exemple Définition de tâche

```
{
  "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"
  ]
}

```

Si vous exécutez votre cluster dans le sous-réseau privé d'un VPC, vous pouvez utiliser le [mode réseau awsvpc](#) pour attacher une interface réseau Elastic (ENI) à vos conteneurs. Cela vous permet d'éviter d'utiliser des liens. Omettez le port hôte dans les mappages de ports, le lien, et la variable d'environnement AWS_XRAY_DAEMON_ADDRESS.

Exemple Définition de tâche de VPC

```

{
  "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
      }
    ]
  }
}
```

Configurer les options de ligne de commande dans la console Amazon ECS

Les options de ligne de commande remplacent les valeurs en conflit dans le fichier de configuration de votre image. Les options de ligne de commande sont généralement utilisées pour les tests locaux, mais peuvent également être utilisées par commodité, lors de la définition de variables d'environnement ou pour contrôler le processus de démarrage.

En ajoutant des options de ligne de commande, vous mettez à jour le Docker CMD qui est transmis au conteneur. Pour plus d'informations, consultez le [guide de référence d'exécution Docker](#).

Définir une option de ligne de commande

1. Ouvrez la console Amazon ECS Classic à l'adresse <https://console.aws.amazon.com/ecs/>.
2. Dans la barre de navigation, choisissez la région qui contient votre définition de tâche.
3. Dans le panneau de navigation, sélectionnez Task Definitions (Définition des tâches).
4. Sur la page Task Definitions, cochez la case située à gauche de la définition de tâche à réviser, puis choisissez Create new revision.
5. Dans la page Create new revision of Task Definition (Créer une nouvelle révision de la définition de tâche) sélectionnez le conteneur.
6. Dans la section ENVIRONMENT (ENVIRONNEMENT) ajoutez votre liste d'options de ligne de commande séparées par des virgules dans le champ Command (Commande).
7. Choisissez Mettre à jour.
8. Vérifiez les informations, puis choisissez Create (Créer).

L'exemple suivant montre comment écrire une option de ligne de commande séparée par des virgules pour l'option `RoleARN`. L'`RoleARN` option suppose le rôle IAM spécifié pour télécharger des segments vers un autre compte.

Exemple

```
--role-arn, arn:aws:iam::123456789012:role/xray-cross-account
```

Pour en savoir plus sur les options de ligne de commande disponibles dans X-Ray, consultez [Configuration du AWS X-Ray daemon](#).

Intégration AWS X-Ray avec d'autres Services AWS

Beaucoup Services AWS proposent différents niveaux d'intégration de X-Ray, notamment l'échantillonnage et l'ajout d'en-têtes aux demandes entrantes, l'exécution du daemon X-Ray et l'envoi automatique de données de suivi à X-Ray. L'intégration à X-Ray peut inclure les éléments suivants :

- Instrumentation active — Demandes entrantes d'échantillons et d'instruments
- Instrumentation passive — Demandes d'instruments qui ont été échantillonnées par un autre service
- Suivi des demandes — Ajoute un en-tête de suivi à toutes les demandes entrantes et le propage en aval
- Outillage — Exécute le démon X-Ray pour recevoir des segments du SDK X-Ray

Note

The X-Ray SDKs inclut des plugins pour une intégration supplémentaire avec Services AWS. Par exemple, vous pouvez utiliser le plug-in Elastic Beanstalk du SDK X-Ray pour Java afin d'ajouter des informations sur l'environnement Elastic Beanstalk qui exécute votre application, notamment le nom et l'ID de l'environnement.

Voici quelques exemples de ceux Services AWS qui sont intégrés à X-Ray :

- [AWS Distro for OpenTelemetry \(ADOT\)](#) — Avec ADOT, les ingénieurs peuvent instrumenter leurs applications une seule fois et envoyer des métriques et des traces corrélées à plusieurs AWS solutions de surveillance, notamment Amazon CloudWatch, Amazon Service et AWS X-Ray Amazon Managed OpenSearch Service for Prometheus.
- [AWS Lambda](#) — Instrumentation active et passive des demandes entrantes sur tous les runtimes. AWS Lambda ajoute deux nœuds à votre carte de trace, l'un pour le AWS Lambda service et l'autre pour la fonction. Lorsque vous activez l'instrumentation, AWS Lambda exécute également le démon X-Ray sur les environnements d'exécution Java et Node.js à utiliser avec le SDK X-Ray.
- [Amazon API Gateway](#) — Instrumentation active et passive. API Gateway utilise des règles d'échantillonnage pour déterminer les demandes à enregistrer et ajoute un nœud pour l'étape de passerelle à votre carte de services.

- [AWS Elastic Beanstalk](#) — Outillage. Elastic Beanstalk inclut le daemon X-Ray sur les plateformes suivantes :
 - Java SE — Configurations 2.3.0 et ultérieures
 - Tomcat — Configurations 2.4.0 et ultérieures
 - Node.js — Configurations 3.2.0 et ultérieures
 - Windows Server : toutes les configurations autres que Windows Server Core publiées après le 9 décembre 2016

Vous pouvez utiliser la console Elastic Beanstalk pour demander à Elastic Beanstalk d'exécuter le daemon sur ces plateformes, ou utiliser l'option dans l'espace de noms. `XRayEnabled`
`aws:elasticbeanstalk:xray`

- [Elastic Load Balancing](#) — Suivi des demandes sur les équilibres de charge des applications. L'Application Load Balancer ajoute l'ID de trace à l'en-tête de la demande avant de l'envoyer à un groupe cible.
- [Amazon EventBridge](#) — Instrumentation passive. Si un service qui publie des événements EventBridge est équipé du SDK X-Ray, les cibles des événements recevront l'en-tête de suivi et pourront continuer à propager l'ID de trace d'origine.
- [Amazon Simple Notification Service](#) — Instrumentation passive. Si un éditeur Amazon SNS trace son client à l'aide du SDK X-Ray, les abonnés peuvent récupérer l'en-tête de suivi et continuer à propager le suivi d'origine auprès de l'éditeur ayant le même identifiant de suivi.
- [Amazon Simple Queue Service](#) — Instrumentation passive. Si un service assure le suivi des demandes à l'aide du SDK X-Ray, Amazon SQS peut envoyer l'en-tête de suivi et continuer à propager le suivi d'origine de l'expéditeur au consommateur avec un identifiant de suivi cohérent.

Choisissez parmi les rubriques suivantes pour découvrir l'ensemble des fonctionnalités intégrées Services AWS.

Rubriques

- [AWS Distro pour et OpenTelemetry AWS X-Ray](#)
- [Support de suivi actif d'Amazon API Gateway pour AWS X-Ray](#)
- [Amazon EC2 et AWS App Mesh](#)
- [AWS App Runner et X-Ray](#)
- [AWS AppSync et AWS X-Ray](#)
- [Enregistrement des appels d'API X-Ray avec AWS CloudTrail](#)

- [CloudWatch intégration avec X-Ray](#)
- [Suivi des modifications de configuration du chiffrement X-Ray avec AWS Config](#)
- [Amazon Elastic Compute Cloud et AWS X-Ray](#)
- [AWS Elastic Beanstalk et AWS X-Ray](#)
- [Elastic Load Balancing et AWS X-Ray](#)
- [Amazon EventBridge et AWS X-Ray](#)
- [AWS Lambda et AWS X-Ray](#)
- [Amazon SNS et AWS X-Ray](#)
- [AWS Step Functions et AWS X-Ray](#)
- [Amazon SQS et AWS X-Ray](#)
- [Amazon S3 et AWS X-Ray](#)

AWS Distro pour et OpenTelemetry AWS X-Ray

Utilisez le AWS Distro for OpenTelemetry (ADOT) pour collecter et envoyer des métriques et des traces à d'autres solutions de surveillance, telles qu'Amazon, CloudWatch Amazon Service AWS X-Ray et Amazon Managed OpenSearch Service for Prometheus.

AWS Distro pour OpenTelemetry

The AWS Distro for OpenTelemetry (ADOT) est une AWS distribution basée sur le projet Cloud Native Computing Foundation (CNCF). OpenTelemetry OpenTelemetry fournit un ensemble unique d'open source APIs, de bibliothèques et d'agents pour collecter des traces et des métriques distribuées. Cette boîte à outils est une distribution de OpenTelemetry composants en amont SDKs, notamment des agents d'auto-instrumentation et des collecteurs testés, optimisés, sécurisés et pris en charge par AWS.

Avec ADOT, les ingénieurs peuvent instrumenter leurs applications une seule fois et envoyer des métriques et des traces corrélées à plusieurs solutions de AWS surveillance, notamment Amazon CloudWatch, AWS X-Ray Amazon OpenSearch Service et Amazon Managed Service for Prometheus.

ADOT est intégré à un nombre croissant de Services AWS afin de simplifier l'envoi de traces et de mesures vers des solutions de surveillance telles que X-Ray. Voici quelques exemples de services intégrés à ADOT :

- AWS Lambda— les couches Lambda AWS gérées pour ADOT offrent une expérience plug-and-play utilisateur en instrumentant automatiquement une fonction Lambda, OpenTelemetry en l'intégrant à une out-of-the-box configuration pour AWS Lambda et X-Ray dans une couche facile à configurer. Les utilisateurs peuvent activer et désactiver OpenTelemetry leur fonction Lambda sans modifier le code. Pour plus d'informations, consultez [AWS Distro for Lambda OpenTelemetry](#)
- Amazon Elastic Container Service (ECS) : collectez des métriques et des traces à partir d'applications Amazon ECS à l'aide de AWS Distro for OpenTelemetry Collector, pour les envoyer à X-Ray et à d'autres solutions de surveillance. Pour plus d'informations, consultez la section [Collecte des données de suivi des applications](#) dans le guide du développeur Amazon ECS.
- AWS App Runner — App Runner prend en charge l'envoi de traces à X-Ray à l'aide de AWS Distro for OpenTelemetry (ADOT). Utilisez ADOT SDKs pour collecter des données de suivi pour vos applications conteneurisées, et utilisez X-Ray pour analyser et obtenir des informations sur votre application instrumentée. Pour plus d'informations, consultez [AWS App Runner et X-Ray](#).

Pour plus d'informations sur la AWS distribution pour OpenTelemetry, y compris l'intégration avec des fonctionnalités supplémentaires Services AWS, consultez la [AWS distribution pour OpenTelemetry](#) la documentation.

Pour plus d'informations sur l'instrumentation de votre application avec AWS Distro for et OpenTelemetry X-Ray, consultez [Instrumenter votre application avec](#) Distro pour. AWS OpenTelemetry

Support de suivi actif d'Amazon API Gateway pour AWS X-Ray

Vous pouvez utiliser X-Ray pour suivre et analyser les demandes des utilisateurs lorsqu'elles transitent par votre Amazon API Gateway APIs pour accéder aux services sous-jacents. API Gateway prend en charge le suivi X-Ray pour tous les types de points de terminaison API Gateway : régionaux, optimisés pour les périphériques et privés. Vous pouvez utiliser X-Ray avec Amazon API Gateway partout Régions AWS où X-Ray est disponible. Pour plus d'informations, consultez [Trace API Gateway API Execution with AWS X-Ray](#) dans le manuel Amazon API Gateway Developer Guide.

Note

X-Ray prend uniquement en charge le suivi pour REST APIs via API Gateway.

Amazon API Gateway fournit un support de [suivi actif](#) pour AWS X-Ray. Activez le suivi actif sur les étapes de votre API pour échantillonner les demandes entrantes et envoyer des traces à X-Ray.

Pour activer le suivi actif sur une étape d'API

1. Ouvrez la console API Gateway à l'adresse <https://console.aws.amazon.com/apigateway/>.
2. Choisissez une API.
3. Choisissez une étape.
4. Dans l'onglet Logs/Tracing, choisissez Enable X-Ray Tracing, puis sélectionnez Save Changes.
5. Dans le volet gauche de navigation, choisissez Ressources.
6. Pour redéployer l'API avec les nouveaux paramètres, choisissez le menu déroulant Actions, puis sélectionnez Déployer l'API.

API Gateway utilise des règles d'échantillonnage que vous définissez dans la console X-Ray pour déterminer les demandes à enregistrer. Vous pouvez créer des règles qui s'appliquent uniquement ou uniquement aux demandes contenant certains en-têtes. API Gateway enregistre les en-têtes dans les attributs du segment, ainsi que des détails sur l'étape et la demande. Pour de plus amples informations, veuillez consulter [Configuration de règles d'échantillonnage](#).

Note

Lors du suivi de l'[intégration HTTP](#) entre REST et API Gateway, le nom de service de chaque segment est défini sur le chemin URL de la demande entre API Gateway et votre point de terminaison d'intégration HTTP, ce qui donne un nœud de service sur la carte de suivi de X-Ray pour chaque chemin d'URL unique. Un grand nombre de chemins d'URL peut faire en sorte que la carte de trace dépasse la limite de 10 000 nœuds, ce qui entraîne une erreur.

Pour minimiser le nombre de nœuds de service créés par API Gateway, pensez à transmettre des paramètres dans la chaîne de requête URL ou dans le corps de la demande via POST. L'une ou l'autre approche garantit que les paramètres ne font pas partie du chemin d'URL, ce qui peut réduire le nombre de chemins d'URL et de nœuds de service distincts.

Pour toutes les demandes entrantes, API Gateway ajoute un [en-tête de suivi](#) aux requêtes HTTP entrantes qui n'en ont pas déjà un.

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

Format d'identification X-Ray Trace

Un X-Ray `trace_id` est composé de trois chiffres séparés par des tirets. Par exemple, `1-58406520-a006649127e371903a2de979`. Cela consiste notamment à :

- Le numéro de version, qui est 1.
- L'heure de la demande d'origine sous Unix Epoch Time en utilisant 8 chiffres hexadécimaux.

Par exemple, le 1er décembre 2016 à 10 h 00 PST est exprimé en 1480615200 secondes ou 58406520 en chiffres hexadécimaux.

- Identifiant 96 bits unique au monde pour la trace en 24 chiffres hexadécimaux.

Si le suivi actif est désactivé, l'étape enregistre toujours un segment si la demande provient d'un service qui a échantillonné la demande et démarré une trace. Par exemple, une application Web instrumentée peut appeler une API API Gateway avec un client HTTP. Lorsque vous instrumentez un client HTTP avec le SDK X-Ray, celui-ci ajoute un en-tête de suivi à la demande sortante contenant la décision d'échantillonnage. API Gateway lit l'en-tête de suivi et crée un segment pour les requêtes échantillonnées.

Si vous utilisez API Gateway pour [générer un SDK Java pour votre API](#), vous pouvez instrumenter le client du SDK en ajoutant un gestionnaire de requêtes avec le générateur de clients, de la même manière que vous instrumentiez manuellement un AWS client SDK. Pour obtenir des instructions, consultez [Suivi des appels du AWS SDK avec le SDK X-Ray pour Java](#).

Amazon EC2 et AWS App Mesh

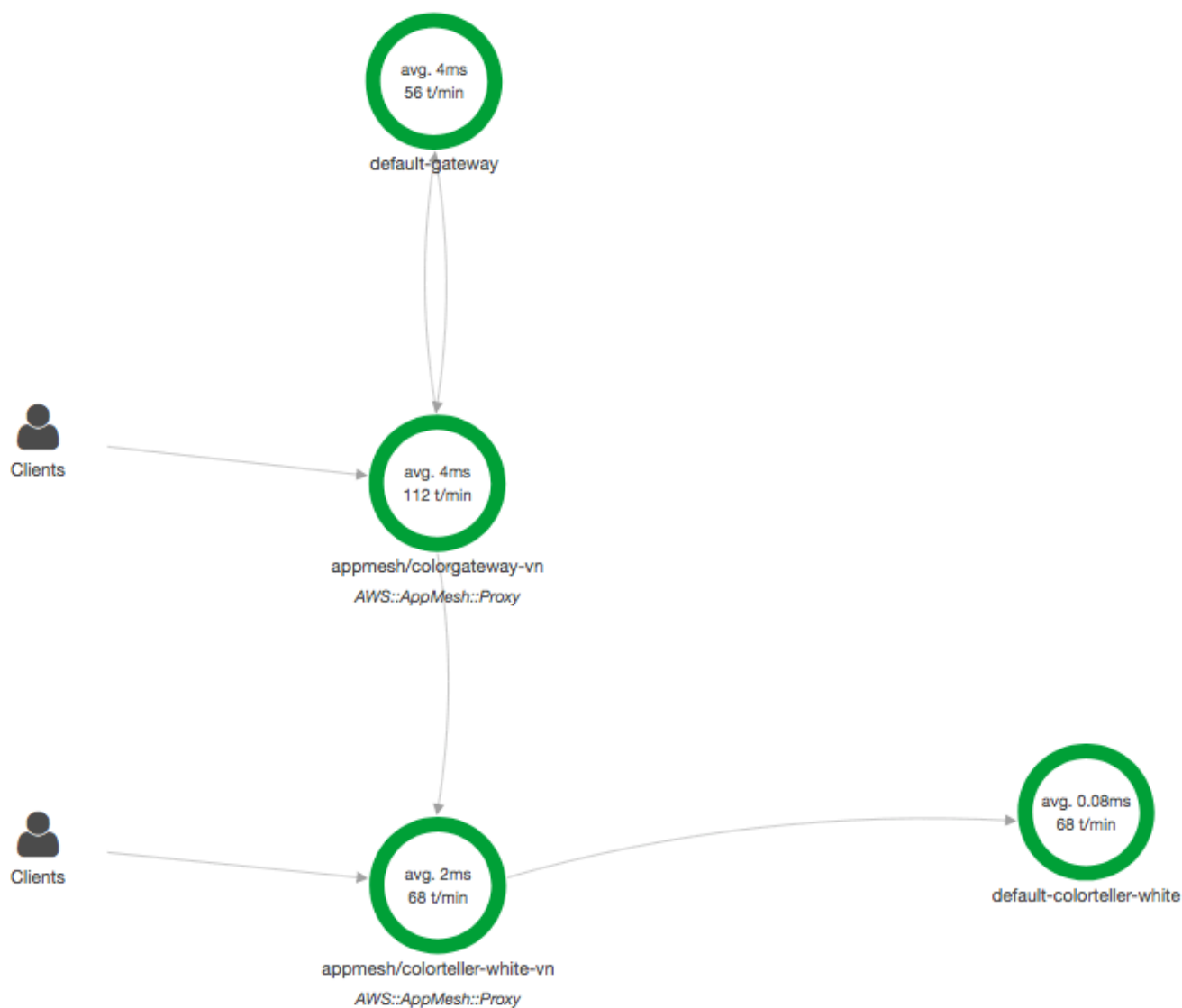
AWS X-Ray s'intègre [AWS App Mesh](#) pour gérer les proxys Envoy pour les microservices. App Mesh fournit une version d'Envoy que vous pouvez configurer pour envoyer des données de suivi au daemon X-Ray exécuté dans un conteneur de la même tâche ou du même pod. X-Ray prend en charge le traçage avec les services compatibles App Mesh suivants :

- Amazon Elastic Container Service (Amazon ECS)
- Amazon Elastic Kubernetes Service (Amazon EKS)
- Amazon Elastic Compute Cloud (Amazon EC2)

Utilisez les instructions suivantes pour apprendre à activer le suivi X-Ray via l'application Mesh.

Service map

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



Pour configurer le proxy Envoy afin qu'il envoie des données à X-Ray, définissez la [variable d'ENABLE_ENVOY_XRAY_TRACING](#) dans sa définition de conteneur.

Note

La version App Mesh d'Envoy n'envoie actuellement pas de traces en fonction des [règles d'échantillonnage](#) configurées. Il utilise plutôt un taux d'échantillonnage fixe de 5 % pour la

version 1.16.3 ou ultérieure d'Envoy, ou un taux d'échantillonnage de 50 % pour les versions d'Envoy antérieures à la version 1.16.3.

Exemple Définition du conteneur Envoy pour Amazon ECS

```
{
  "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

Pour en savoir plus sur les adresses régionales Envoy disponibles, consultez [l'image d'Envoy](#) dans le guide de AWS App Mesh l'utilisateur.

Pour plus de détails sur l'exécution du démon X-Ray dans un conteneur, consultez [Exécution du daemon X-Ray sur Amazon ECS](#). Pour un exemple d'application qui inclut un service mesh, un microservice, un proxy Envoy et un daemon X-Ray, déployez l'`colorappexemple` dans le référentiel [App Mesh Examples GitHub](#).

En savoir plus

- [Mise en route avec AWS App Mesh](#)
- [Mise en route avec Amazon ECS AWS App Mesh et Amazon ECS](#)

AWS App Runner et X-Ray

AWS App Runner est un Service AWS outil qui fournit un moyen rapide, simple et rentable de déployer directement à partir du code source ou d'une image de conteneur vers une application Web évolutive et sécurisée dans le AWS Cloud. Vous n'avez pas besoin d'apprendre de nouvelles technologies, de choisir le service informatique à utiliser ou de savoir comment approvisionner et configurer les AWS ressources. Consultez [Qu'est-ce qu' AWS App Runner](#) pour plus d'informations.

AWS App Runner envoie des traces à X-Ray en les intégrant à [AWS Distro for OpenTelemetry](#) (ADOT). Utilisez ADOT SDKs pour collecter des données de suivi pour vos applications conteneurisées, et utilisez X-Ray pour analyser et obtenir des informations sur votre application instrumentée. Pour plus d'informations, consultez la section [Suivi de votre application App Runner avec X-Ray](#).

AWS AppSync et AWS X-Ray

Vous pouvez activer et suivre les demandes pour AWS AppSync. Pour plus d'informations, voir [Tracing with AWS X-Ray](#) pour obtenir des instructions.

Lorsque le suivi X-Ray est activé pour une AWS AppSync API, un [rôle lié au service AWS](#) Identity and Access Management est automatiquement créé dans votre compte avec les autorisations appropriées. Cela permet AWS AppSync d'envoyer des traces à X-Ray de manière sécurisée.

Enregistrement des appels d'API X-Ray avec AWS CloudTrail

AWS X-Ray est intégré à [AWS CloudTrail](#) un service qui fournit un enregistrement des actions entreprises par un utilisateur, un rôle ou un Service AWS. CloudTrail capture tous les appels d'API pour X-Ray sous forme d'événements. Les appels capturés incluent des appels provenant de la console X-Ray et des appels de code vers les opérations de l'API X-Ray. À l'aide des informations collectées par CloudTrail, vous pouvez déterminer la demande qui a été faite à X-Ray, l'adresse IP à partir de laquelle la demande a été faite, la date à laquelle elle a été faite et des informations supplémentaires.

Chaque événement ou entrée de journal contient des informations sur la personne ayant initié la demande. Les informations relatives à l'identité permettent de déterminer :

- Si la demande a été effectuée avec des informations d'identification d'utilisateur root ou d'utilisateur root.
- Si la demande a été faite au nom d'un utilisateur du centre d'identité IAM.
- Si la demande a été effectuée avec les informations d'identification de sécurité temporaires d'un rôle ou d'un utilisateur fédéré.
- Si la requête a été effectuée par un autre Service AWS.

CloudTrail est actif dans votre compte Compte AWS lorsque vous créez le compte et vous avez automatiquement accès à l'historique des CloudTrail événements. L'historique des CloudTrail événements fournit un enregistrement consultable, consultable, téléchargeable et immuable des 90 derniers jours des événements de gestion enregistrés dans un. Région AWS Pour plus d'informations, consultez la section [Utilisation de l'historique des CloudTrail événements](#) dans le guide de AWS CloudTrail l'utilisateur. La consultation de CloudTrail l'historique des événements est gratuite.

Pour un enregistrement continu des événements de vos 90 Compte AWS derniers jours, créez un magasin de données sur les événements de Trail ou [CloudTrailLake](#).

CloudTrail sentiers

Un suivi permet CloudTrail de fournir des fichiers journaux à un compartiment Amazon S3. Tous les sentiers créés à l'aide du AWS Management Console sont multirégionaux. Vous ne pouvez créer un journal de suivi en une ou plusieurs régions à l'aide de l' AWS CLI. Il est recommandé de créer un parcours multirégional, car vous capturez l'activité dans l'ensemble Régions AWS de votre compte. Si vous créez un journal de suivi pour une seule région, il convient de n'afficher que les événements enregistrés dans le journal de suivi pour une seule région Région AWS. Pour plus d'informations sur les journaux de suivi, consultez [Créez un journal de suivi dans vos Compte AWS](#) et [Création d'un journal de suivi pour une organisation](#) dans le AWS CloudTrail Guide de l'utilisateur.

Vous pouvez envoyer une copie de vos événements de gestion en cours à votre compartiment Amazon S3 gratuitement CloudTrail en créant un journal. Toutefois, des frais de stockage Amazon S3 sont facturés. Pour plus d'informations sur la CloudTrail tarification, consultez la section [AWS CloudTrail Tarification](#). Pour obtenir des informations sur la tarification Amazon S3, consultez [Tarification Amazon S3](#).

CloudTrail Stockages de données sur les événements du lac

CloudTrail Lake vous permet d'exécuter des requêtes SQL sur vos événements. CloudTrail Lake convertit les événements existants au format JSON basé sur les lignes au format [Apache ORC](#). ORC est un format de stockage en colonnes qui est optimisé pour une récupération rapide des données. Les événements sont agrégés dans des magasins de données d'événement. Ceux-ci constituent des collections immuables d'événements basées sur des critères que vous sélectionnez en appliquant des [sélecteurs d'événements avancés](#). Les sélecteurs que vous appliquez à un magasin de données d'événement contrôlent les événements qui persistent et que vous pouvez interroger. Pour plus d'informations sur CloudTrail Lake, consultez la section [Travailler avec AWS CloudTrail Lake](#) dans le guide de AWS CloudTrail l'utilisateur.

CloudTrail Les stockages et requêtes de données sur les événements de Lake entraînent des coûts. Lorsque vous créez un magasin de données d'événement, vous choisissez l'[option de tarification](#) que vous voulez utiliser pour le magasin de données d'événement. L'option de tarification détermine le coût d'ingestion et de stockage des événements, ainsi que les périodes de conservation par défaut et maximale pour le magasin de données d'événement. Pour plus d'informations sur la CloudTrail tarification, consultez la section [AWS CloudTrail Tarification](#).

Rubriques

- [Événements relatifs à la gestion des rayons X dans CloudTrail](#)
- [Événements liés aux données X-Ray dans CloudTrail](#)
- [Exemples d'événements X-Ray](#)

Événements relatifs à la gestion des rayons X dans CloudTrail

AWS X-Ray s'intègre AWS CloudTrail pour enregistrer les actions d'API effectuées par un utilisateur, un rôle ou un Service AWS in X-Ray. Vous pouvez l'utiliser CloudTrail pour surveiller les demandes d'API X-Ray en temps réel et stocker les journaux dans Amazon S3, Amazon CloudWatch Logs et Amazon CloudWatch Events. X-Ray prend en charge l'enregistrement des actions suivantes sous forme d'événements dans des fichiers CloudTrail journaux :

Actions d'API prises en charge

- [PutEncryptionConfig](#)
- [GetEncryptionConfig](#)

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

Événements liés aux données X-Ray dans CloudTrail

[Les événements de données](#) fournissent des informations sur les opérations de ressources effectuées sur ou dans une ressource (par exemple, [PutTraceSegments](#), qui télécharge les documents segmentés vers X-Ray).

Ils sont également connus sous le nom opérations de plans de données. Les événements de données sont souvent des activités dont le volume est élevé. Par défaut, CloudTrail n'enregistre pas les événements liés aux données. L'historique des CloudTrail événements n'enregistre pas les événements liés aux données.

Des frais supplémentaires s'appliquent pour les événements de données. Pour plus d'informations sur la CloudTrail tarification, consultez la section [AWS CloudTrail Tarification](#).

Vous pouvez enregistrer les événements de données pour les types de ressources X-Ray à l'aide de la CloudTrail console ou AWS CLI des opérations de CloudTrail l'API. Pour plus d'informations sur la façon de journaliser les événements de données, consultez [Journalisation des événements de données avec la AWS Management Console](#) et [Journalisation des événements de données avec l'AWS Command Line Interface](#) dans le Guide de l'utilisateur AWS CloudTrail .

Le tableau suivant répertorie les types de ressources X-Ray pour lesquels vous pouvez enregistrer des événements de données. La colonne Type d'événement de données (console) indique la valeur à choisir dans la liste des types d'événements de données de la CloudTrail console. La colonne de valeur `resources.type` indique la **resources.type** valeur que vous devez spécifier lors de la

configuration de sélecteurs d'événements avancés à l'aide du ou. AWS CLI CloudTrail APIs La CloudTrail colonne Données APIs enregistrées indique les appels d'API enregistrés CloudTrail pour le type de ressource.

Type d'événement de données (console)	valeur resources.type	Données APIs enregistrées sur CloudTrail
X-Ray Trace	AWS::XRay::Trace	• PutTraceSegments • GetTraceSummaries • GetTraceGraph • GetServiceGraph • BatchGetTraces • GetTimeSeriesServiceStatistics • PutTelemetryRecords • GetSamplingTargets

Vous pouvez configurer des sélecteurs d'événements avancés pour filtrer les événements eventName et readOnly des champs pour n'enregistrer que les événements importants pour vous. Cependant, vous ne pouvez pas sélectionner d'événements en ajoutant le sélecteur de resources.ARN champs, car X-Ray Traces n'en possède ARNs pas. Pour plus d'informations sur ces champs, voir [AdvancedFieldSelector](#) dans la Référence d'API AWS CloudTrail. Voici un exemple de la manière d'exécuter la [put-event-selectors](#) AWS CLI commande pour enregistrer les événements liés aux données lors d'un CloudTrail suivi. Vous devez exécuter la commande ou spécifier la région dans laquelle le suivi a été créé ; sinon, l'opération renvoie une InvalidHomeRegionException exception.

```
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"
  }
]
}'

```

Exemples d'événements X-Ray

Exemple d'événement de gestion, **GetEncryptionConfig**

Voici un exemple du X-Ray GetEncryptionConfig connectez-vous à votre compte CloudTrail.

Exemple

```

{
  "eventVersion"=>"1.05",
  "userIdentity"=>{
    "type"=>"AssumedRole",
    "principalId"=>"AR0AJVHBZWD3DN6CI2MHM: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"=>"AR0AJVHBZWD3DN6CI2MHM",
        "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"
  }

```

Exemple d'événement lié aux données, **PutTraceSegments**

Voici un exemple du X-Ray PutTraceSegments entrée du journal des événements de données dans CloudTrail.

Exemple

```

{
  "eventVersion": "1.09",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AROAWYXPW54Y4NEXAMPLE:i-0dzz2ac111c83zz0z",
    "arn": "arn:aws:sts::012345678910:assumed-role/my-service-role/i-0dzz2ac111c83zz0z",
    "accountId": "012345678910",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AROAWYXPW54Y4NEXAMPLE",
        "arn": "arn:aws:iam::012345678910:role/service-role/my-service-role",
        "accountId": "012345678910",
        "userName": "my-service-role"
      },
      "attributes": {
        "creationDate": "2024-01-22T17:34:11Z",
        "mfaAuthenticated": "false"
      },
      "ec2RoleDelivery": "2.0"
    }
  },
  "eventTime": "2024-01-22T18:22:05Z",
  "eventSource": "xray.amazonaws.com",
  "eventName": "PutTraceSegments",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "198.51.100.0",

```



```

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

```

CloudWatch intégration avec X-Ray

AWS X-Ray s'intègre à [CloudWatch Application Signals](#), CloudWatch RUM et CloudWatch Synthetics pour faciliter le suivi de l'état de vos applications. Activez votre application pour Application Signals afin de surveiller et de résoudre les problèmes liés à l'état de fonctionnement de vos services, de vos pages clients, des canaries Synthetics et des dépendances des services.

En corrélant CloudWatch les métriques, les journaux et les traces X-Ray, la carte de suivi X-Ray fournit une end-to-end vue de vos services pour vous aider à identifier rapidement les problèmes de performance et à identifier les utilisateurs concernés.

Avec CloudWatch RUM, vous pouvez effectuer une surveillance réelle des utilisateurs afin de collecter et de visualiser les données côté client concernant les performances de votre application Web à partir de sessions utilisateur réelles en temps quasi réel. Avec AWS X-Ray et CloudWatch RUM, vous pouvez analyser et déboguer le chemin de la demande en commençant par les utilisateurs finaux de votre application via les services AWS gérés en aval. Cela vous permet d'identifier les tendances de latence et les erreurs qui ont un impact sur les utilisateurs finaux.

Rubriques

- [CloudWatch RUM et AWS X-Ray](#)
- [Déboguer des canaris CloudWatch synthétiques à l'aide de X-Ray](#)

CloudWatch RUM et AWS X-Ray

Avec Amazon CloudWatch RUM, vous pouvez effectuer une surveillance réelle des utilisateurs afin de collecter et de consulter les données côté client concernant les performances de votre application Web à partir de sessions utilisateur réelles en temps quasi réel. Avec AWS X-Ray et CloudWatch RUM, vous pouvez analyser et déboguer le chemin de la demande en commençant par les utilisateurs finaux de votre application via les services AWS gérés en aval. Cela vous permet d'identifier les tendances de latence et les erreurs qui ont un impact sur les utilisateurs finaux.

Une fois que vous avez activé le suivi des sessions utilisateur par X-Ray, CloudWatch RUM ajoute un en-tête de trace X-Ray aux requêtes HTTP autorisées et enregistre un segment X-Ray pour les requêtes HTTP autorisées. Vous pouvez ensuite voir les traces et les segments de ces sessions utilisateur dans le X-Ray et les CloudWatch consoles, y compris la carte de suivi X-Ray.

Note

CloudWatch RUM ne s'intègre pas aux règles d'échantillonnage X-Ray. Choisissez plutôt un pourcentage d'échantillonnage lorsque vous configurez votre application pour utiliser CloudWatch RUM. Les traces envoyées depuis CloudWatch RUM peuvent entraîner des frais supplémentaires. Pour en savoir plus, consultez [Pricing AWS X-Ray](#) (Tarification).

Par défaut, les traces côté client envoyées depuis CloudWatch RUM ne sont pas connectées aux traces côté serveur. Pour connecter les traces côté client aux traces côté serveur, configurez le client Web CloudWatch RUM pour ajouter un en-tête de trace X-Ray à ces requêtes HTTP.

Warning

La configuration du client Web CloudWatch RUM pour ajouter un en-tête de trace X-Ray aux requêtes HTTP peut entraîner l'échec du partage de ressources entre origines (CORS). Pour éviter cela, ajoutez l'en-tête X-Amzn-Trace-Id HTTP à la liste des en-têtes autorisés dans la configuration CORS de votre service en aval. Si vous utilisez API Gateway en aval, consultez la section [Activation de CORS pour une ressource d'API REST](#). Nous vous recommandons fortement de tester votre application avant d'ajouter un en-tête de suivi X-Ray côté client dans un environnement de production. Pour plus d'informations, consultez la [documentation du client Web CloudWatch RUM](#).

Pour plus d'informations sur la surveillance des utilisateurs réels dans CloudWatch, voir [Utiliser CloudWatch RUM](#). Pour configurer votre application pour utiliser CloudWatch RUM, y compris le suivi des sessions utilisateur avec X-Ray, voir [Configurer une application pour utiliser CloudWatch RUM](#).

Déboguer des canaris CloudWatch synthétiques à l'aide de X-Ray

CloudWatch Synthetics est un service entièrement géré qui vous permet de surveiller vos terminaux APIs et d'utiliser des canaries scriptés qui s'exécutent 24 heures par jour, une fois par minute.

Vous pouvez personnaliser les scripts canari pour vérifier les changements en termes de :

- Disponibilité
- Latence
- Transactions
- Liens brisés ou morts
- Step-by-step achèvement des tâches
- Erreurs de chargement de page
- Latences de chargement pour les actifs de l'interface utilisateur
- Flux d'assistants complexes
- Flux de caisse dans votre application

Les scripts canaris suivent les mêmes itinéraires et effectuent les mêmes actions et comportements que vos clients, et vérifient continuellement l'expérience client.

Pour en savoir plus sur la configuration des tests Synthetic, consultez [Utilisation des synthétiques pour créer et gérer des scripts canaris](#).



Les exemples suivants montrent des cas d'utilisation courants pour les problèmes de débogage que vos scripts canaris Synthetic soulèvent. Chaque exemple illustre une stratégie clé pour le débogage à l'aide de la carte de trace ou de la console X-Ray Analytics.

Pour plus d'informations sur la façon de lire et d'interagir avec la carte de trace, consultez la section [Affichage de la carte des services](#).

Pour plus d'informations sur la façon de lire et d'interagir avec la console X-Ray Analytics, consultez [Interaction avec la console AWS X-Ray Analytics](#).

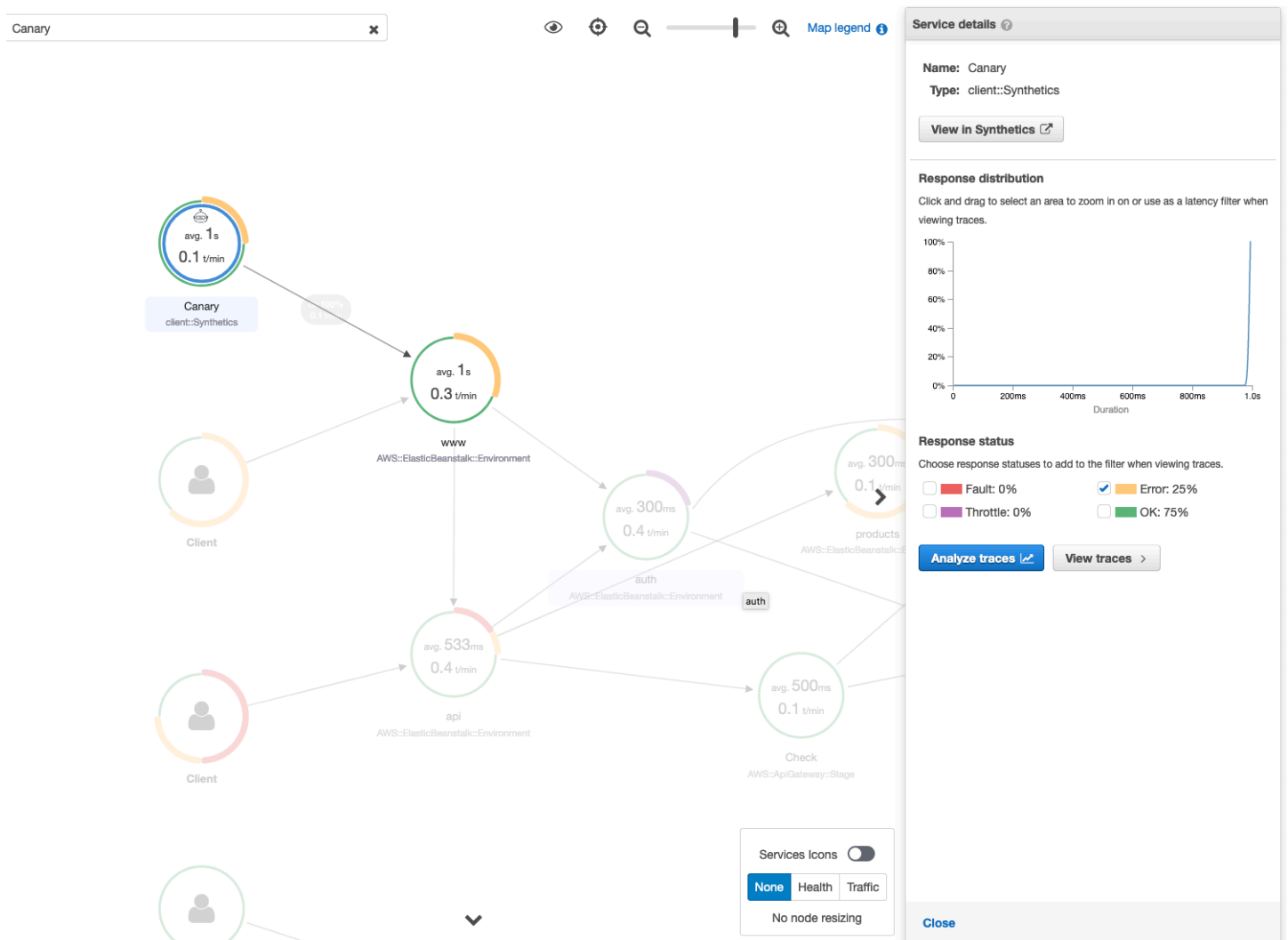
Rubriques

- [Afficher les canaris avec un nombre accru de rapports d'erreurs sur la carte de suivi](#)
- [Utilisez des cartes détaillées pour les traces individuelles afin de visualiser chaque demande en détail](#)
- [Détermination de la cause racine des défaillances en cours dans les services en amont et en aval](#)
- [Identification des goulets d'étranglement et les tendances en matière de performances](#)
- [Comparaison des taux de latence et d'erreur ou d'erreur avant et après les modifications](#)
- [Déterminer la couverture canarienne requise pour tous APIs et URLs](#)
- [Utilisation des groupes pour se concentrer sur les tests synthétiques](#)

Afficher les canaris avec un nombre accru de rapports d'erreurs sur la carte de suivi

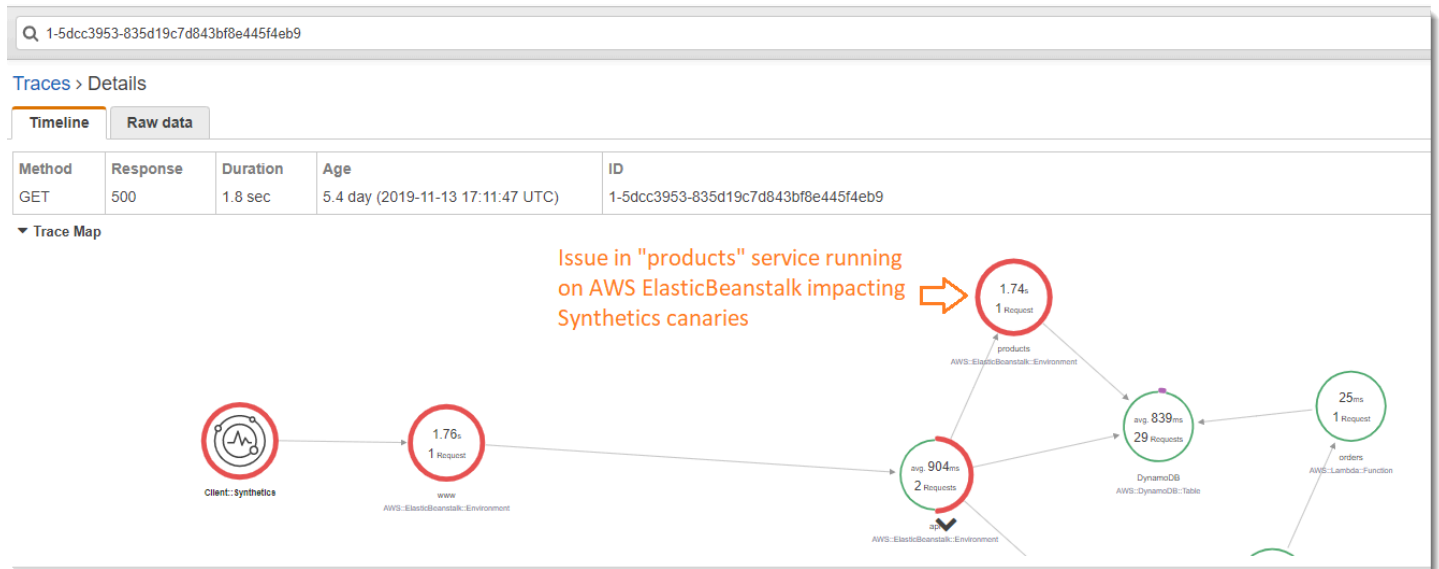
[Pour voir quels canaris présentent une augmentation du nombre d'erreurs, de défaillances, de taux de ralentissement ou de temps de réponse lents sur votre carte de suivi X-Ray, vous pouvez mettre en évidence les nœuds clients Synthetics Canary à l'aide du filtre. Client::Synthetic](#) Cliquez sur un nœud pour afficher la distribution du temps de réponse de l'ensemble de la demande. Cliquez sur une arête entre deux nœuds pour afficher des informations détaillées sur les demandes ayant transité par cette connexion. Vous pouvez également afficher les nœuds inférés « distants » pour les services en aval associés sur votre carte de trace.

Lorsque vous cliquez sur le nœud Synthetics, un bouton Afficher dans Synthetics se trouve sur le panneau latéral qui vous redirige vers la console Synthetics où vous pouvez consulter les détails de Canary.



Utilisez des cartes détaillées pour les traces individuelles afin de visualiser chaque demande en détail

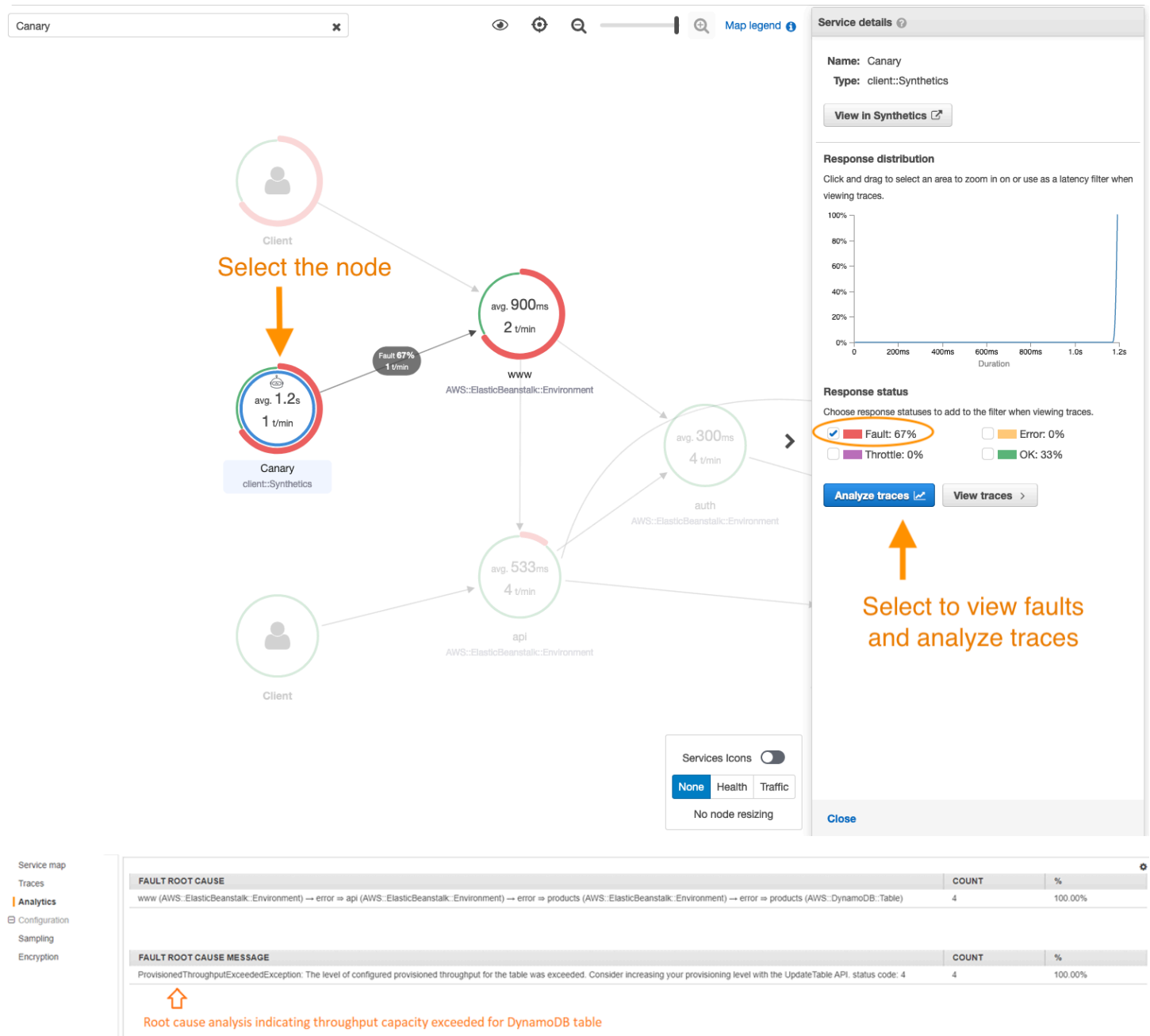
Pour déterminer quel service entraîne le plus de latence ou est à l'origine d'une erreur, appelez la carte des détails du suivi en sélectionnant le tracé dans la carte de suivi. Les cartes détaillées des traces individuelles indiquent le end-to-end chemin d'une seule demande. Utilisez cette option pour comprendre les services invoqués et visualiser les services en amont et en aval.



Détermination de la cause racine des défaillances en cours dans les services en amont et en aval

Une fois que vous avez reçu une CloudWatch alarme signalant une défaillance d'un Synthetics Canary, utilisez la modélisation statistique des données de trace dans X-Ray pour déterminer la cause première probable du problème dans la console X-Ray Analytics. Dans la console Analytics, le tableau Response Time Root Cause indique les chemins d'entités enregistrés. X-Ray détermine la trajectoire de votre trace qui est la cause la plus probable du temps de réponse. Le format indique une hiérarchie d'entités qui a été rencontrée, qui constitue la cause racine du temps de réponse.

L'exemple suivant montre que le test Synthetics pour l'API « XXX » exécuté sur API Gateway échoue en raison d'une exception de capacité de débit provenant de la table Amazon DynamoDB.



AWS:CANARY_ARN	COUNT
arn:aws:synthetics:us-east-1:779168132807:canary:www-test	118

Annotation.acl_cached

Annotation.authenticated

Annotation.aws.canary_arn

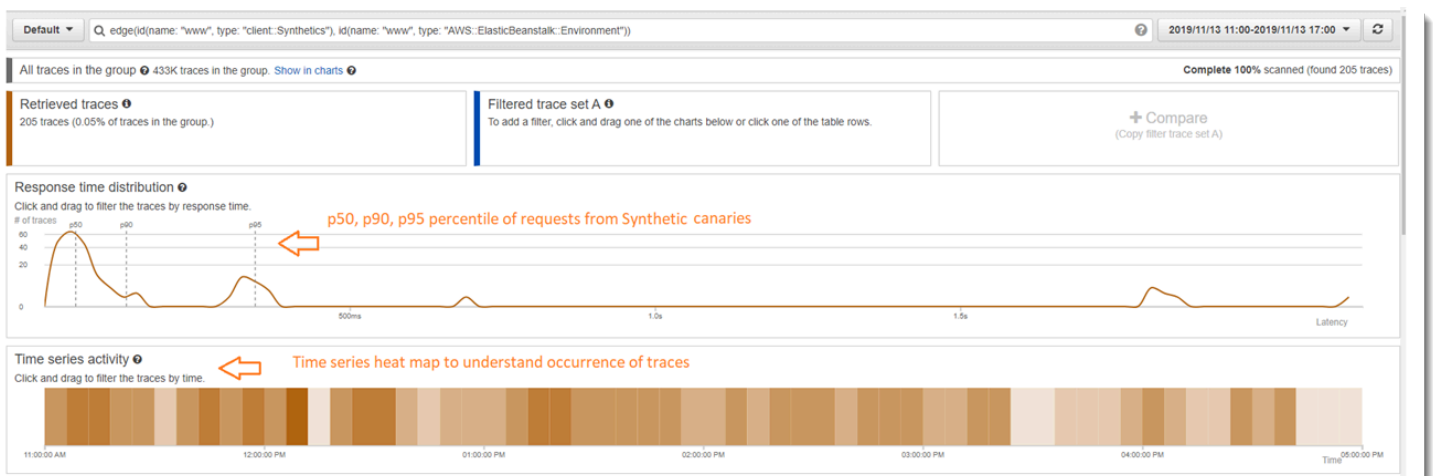
Annotation.cold_start

Annotation.credentials_cached

Annotation.queries

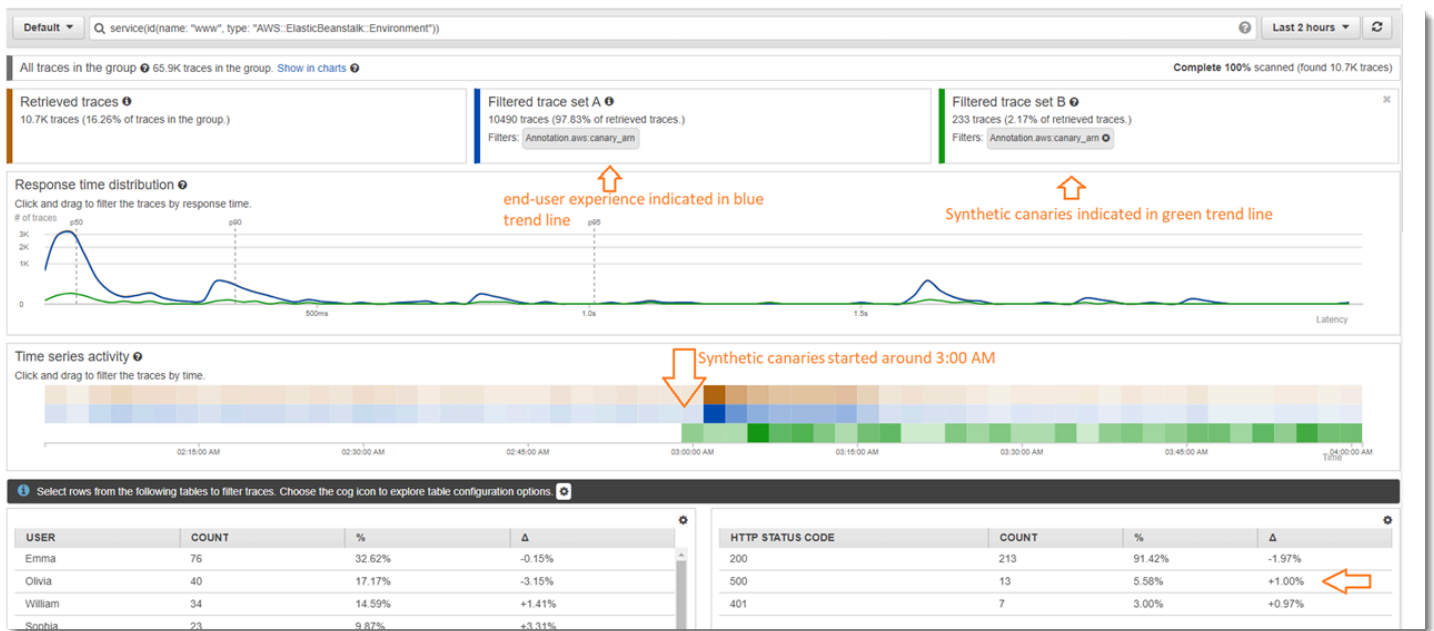
Identification des goulets d'étranglement et les tendances en matière de performances

Vous pouvez visualiser les tendances des performances de votre terminal au fil du temps en utilisant le trafic continu provenant de vos canaris Synthetics pour remplir une carte des détails des traces sur une période donnée.



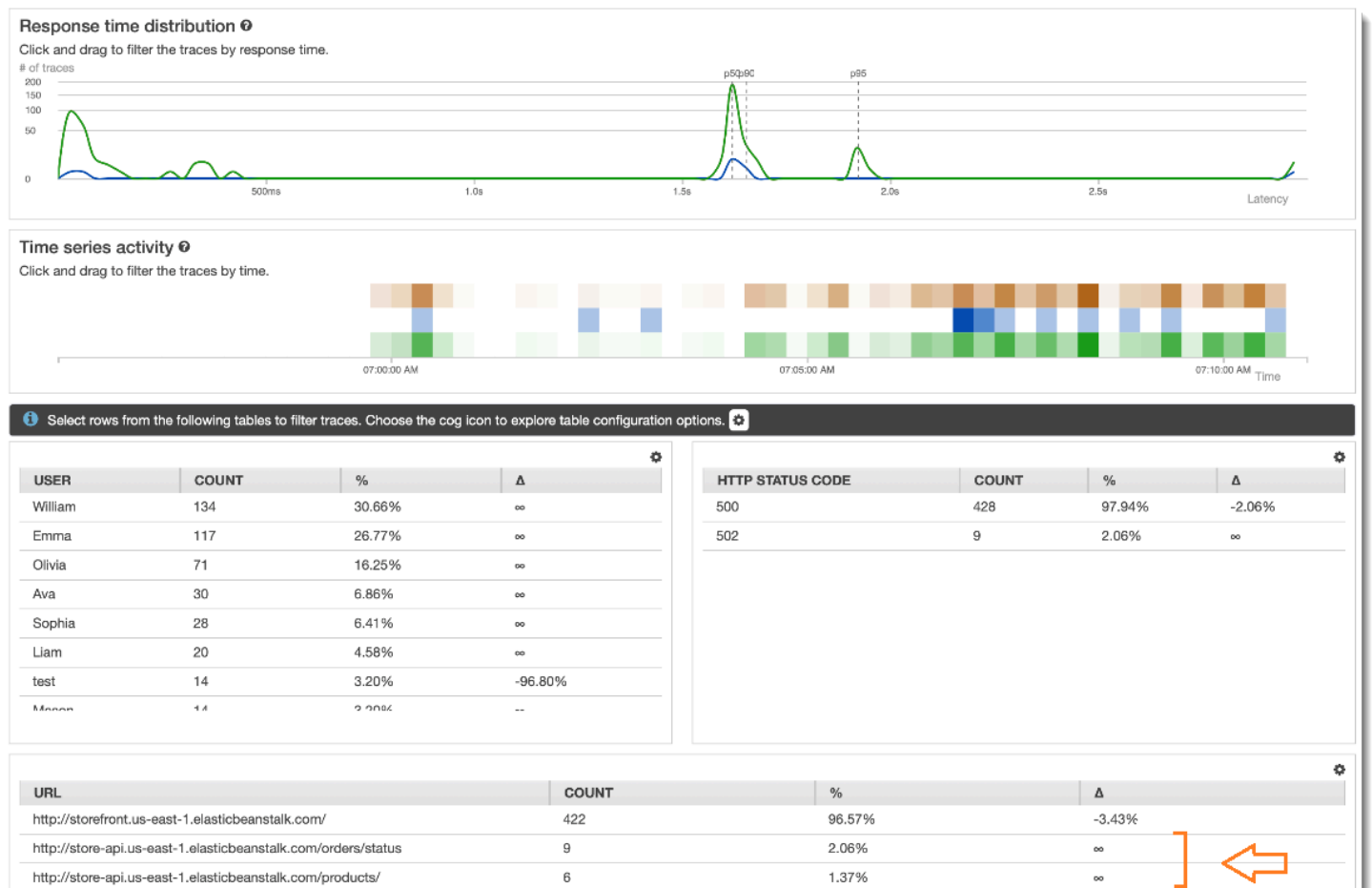
Comparaison des taux de latence et d'erreur ou d'erreur avant et après les modifications

Pincez l'heure à laquelle un changement s'est produit pour corrélérer ce changement à une augmentation du nombre de problèmes détectés par vos canaris. Utilisez la console X-Ray Analytics pour définir les plages de temps avant et après comme des ensembles de traces différents, ce qui permet de différencier visuellement la distribution des temps de réponse.



Déterminer la couverture canarienne requise pour tous APIs et URLs

Utilisez X-Ray Analytics pour comparer l'expérience des canarys avec les utilisateurs. L'interface utilisateur ci-dessous montre une ligne de tendance bleue pour les canarys et une ligne verte pour les utilisateurs. Vous pouvez également identifier que deux d'entre eux URLs ne sont pas soumis au test Canary.

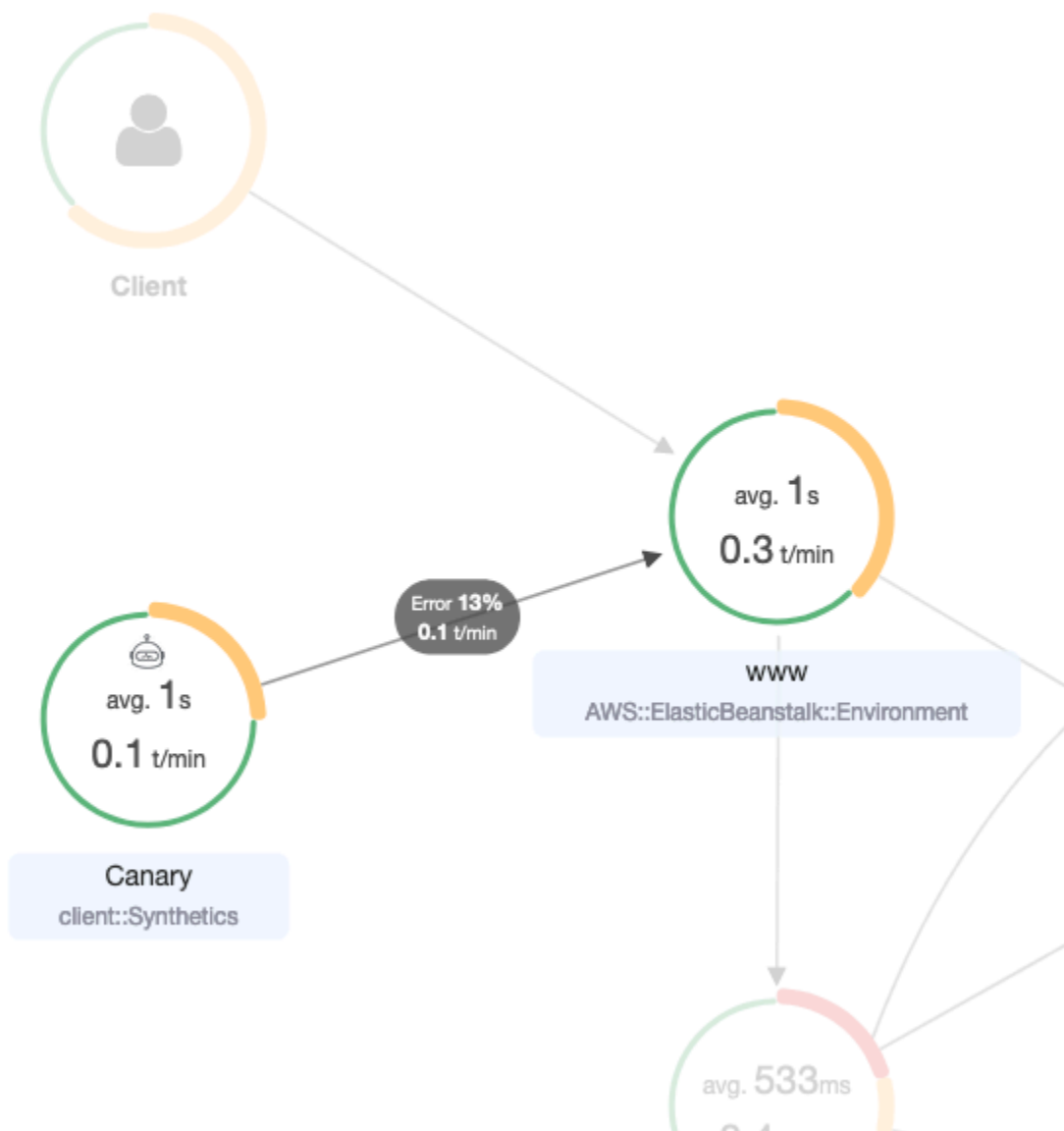


Utilisation des groupes pour se concentrer sur les tests synthétiques

Vous pouvez créer un groupe X-Ray à l'aide d'une expression de filtre pour vous concentrer sur un certain ensemble de flux de travail, tels que les tests Synthetics pour l'application « www » en cours d'exécution. AWS Elastic Beanstalk Utilisez des [mots clés complexes](#) `service()` et `edge()` filtrez les services et les limites.

Exemple Expression de filtre de groupe

```
"edge(id(name: "www", type: "client::Synthetics"), id(name: "www", type:
"AWS::ElasticBeanstalk::Environment"))"
```



Suivi des modifications de configuration du chiffrement X-Ray avec AWS Config

AWS X-Ray s'intègre AWS Config pour enregistrer les modifications de configuration apportées à vos ressources de chiffrement X-Ray. Vous pouvez l'utiliser AWS Config pour inventorier les ressources de chiffrement de X-Ray, vérifier l'historique de configuration de X-Ray et envoyer des notifications en fonction des modifications apportées aux ressources.

AWS Config prend en charge l'enregistrement des modifications des ressources de chiffrement X-Ray suivantes sous forme d'événements :

- Modifications de configuration : modification ou ajout d'une clé de chiffrement, ou retour au paramètre de chiffrement par défaut de X-Ray.

Suivez les instructions ci-dessous pour savoir comment créer une connexion de base entre X-Ray et AWS Config.

Création d'un déclencheur de fonction Lambda

Vous devez disposer de l'ARN d'une AWS Lambda fonction personnalisée avant de pouvoir générer une AWS Config règle personnalisée. Suivez ces instructions pour créer une fonction élémentaire avec Node.js qui renvoie à AWS Config une valeur conforme ou non conforme en fonction de l'état de la ressource `XrayEncryptionConfig`.

Pour créer une fonction Lambda avec un `AWS::Xray EncryptionConfig` déclencheur de modification

1. Ouvrez la [console Lambda](#). Sélectionnez Créer une fonction.
2. Choisissez Blueprints, puis filtrez la bibliothèque de plans pour le `config-rule-change-triggeredplan`. Cliquez sur le lien dans le plan ou choisissez Configurer (Configurer) pour continuer.
3. Définissez les champs suivants pour configurer le plan :
 - Pour Name (Nom), tapez un nom.
 - Pour Role (Rôle), choisissez Create new role from template(s) (Créer un rôle à partir d'un modèle).
 - Tapez un nom pour Role name (Nom de rôle).
 - Pour Policy templates (Modèles de stratégie), choisissez AWS Config Rules permissions (Autorisations de règles &CC;).
4. Choisissez Créer une fonction pour créer et afficher votre fonction dans la AWS Lambda console.
5. Modifiez le code de votre fonction pour remplacer `AWS::EC2::Instance` par `AWS::XrayEncryptionConfig`. Vous pouvez également mettre à jour le champ de description pour refléter cette modification.

Code par défaut

```
if (configurationItem.resourceType !== 'AWS::EC2::Instance') {  
    return 'NOT_APPLICABLE';  
}
```

```

    } else if (ruleParameters.desiredInstanceType ===
configurationItem.configuration.instanceType) {
        return 'COMPLIANT';
    }
    return 'NON_COMPLIANT';

```

Code mis à jour

```

    if (configurationItem.resourceType !== 'AWS::XRay::EncryptionConfig') {
        return 'NOT_APPLICABLE';
    } else if (ruleParameters.desiredInstanceType ===
configurationItem.configuration.instanceType) {
        return 'COMPLIANT';
    }
    return 'NON_COMPLIANT';

```

6. Ajoutez ce qui suit à votre rôle d'exécution dans IAM pour accéder à X-Ray. Ces autorisations permettent un accès en lecture seule à vos ressources X-Ray. Le fait de ne pas fournir l'accès aux ressources appropriées entraînera un message hors de portée AWS Config dès l'évaluation de la fonction Lambda associée à la règle.

```

{
    "Sid": "Stmt1529350291539",
    "Action": [
        "xray:GetEncryptionConfig"
    ],
    "Effect": "Allow",
    "Resource": "*"
}

```

Création d'une AWS Config règle personnalisée pour la radiographie

Lorsque la fonction Lambda est créée, notez son ARN et accédez à la AWS Config console pour créer votre règle personnalisée.

Pour créer une AWS Config règle pour X-Ray

1. Ouvrez la page [Rules \(Règles\) de la console AWS Config](#).
2. Choisissez Add rule (Ajouter une règle), puis Add custom rule (Ajouter une règle personnalisée).

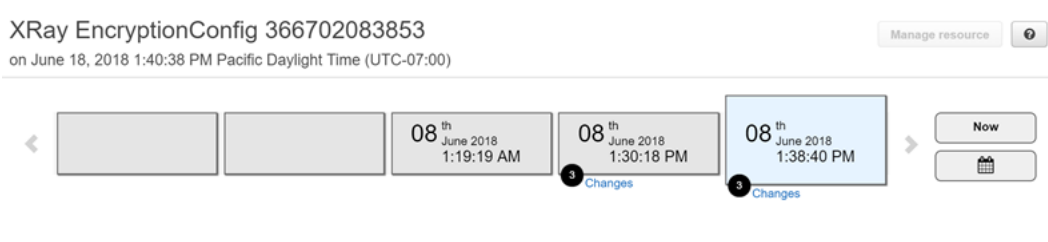
3. Dans AWS Lambda Function ARN, insérez l'ARN associé à la fonction Lambda que vous souhaitez utiliser.
4. Choisissez le type de déclencheur à définir :
 - Modifications de configuration : AWS Config déclenche l'évaluation lorsqu'une ressource correspondant au champ d'application de la règle change de configuration. L'évaluation s'exécute après l' AWS Config envoi d'une notification de modification d'élément de configuration.
 - Périodique : AWS Config exécute des évaluations de la règle à la fréquence que vous choisissez (par exemple, toutes les 24 heures).
5. Pour Resource type (Type de ressource), choisissez EncryptionConfig dans la section X-Ray.
6. Choisissez Save (Enregistrer).

La AWS Config console commence immédiatement à évaluer la conformité de la règle. L'évaluation peut prendre plusieurs minutes.

Maintenant que cette règle est conforme, AWS Config vous pouvez commencer à compiler un historique d'audit. AWS Config enregistre les modifications des ressources sous forme de chronologie. Pour chaque modification de la chronologie des événements, AWS Config génère un tableau au format fro/to pour montrer ce qui a changé dans la représentation JSON de la clé de chiffrement. Les deux modifications de champ associées à EncryptionConfig sont `Configuration.type` et `Configuration.keyID`.

Exemples de résultat

Voici un exemple de AWS Config chronologie montrant les modifications apportées à des dates et heures spécifiques.



Voici un exemple d'entrée de AWS Config modification. Le format de/à illustre ce qui a été modifié. Cet exemple montre que les paramètres de chiffrement par défaut de X-Ray ont été remplacés par une clé de chiffrement définie.

▼ Changes 3

Configuration Changes 3

Field	From	To
SupplementaryConfiguration.unsupportedResources		<pre> Array [1] 0: Object resourceId: "arn:aws:kms:us-west-2:366702083853:key/e0531084-06ad-4d7f-9ea6-53dd693a945c" resourceType: "AWS::KMS::Key" </pre>
Configuration.type	"NONE"	"KMS"
Configuration.keyId		"arn:aws:kms:us-west-2:366702083853:key/e0531084-06ad-4d7f-9ea6-53dd693a945c"

Notifications Amazon SNS

Pour être informé des modifications de configuration, configurez AWS Config pour publier les notifications Amazon SNS. Pour plus d'informations, consultez la section [Surveillance AWS Config des modifications des ressources par e-mail](#).

Amazon Elastic Compute Cloud et AWS X-Ray

Vous pouvez installer et exécuter le daemon X-Ray sur une EC2 instance Amazon à l'aide d'un script de données utilisateur. Pour obtenir des instructions, consultez [Exécution du daemon X-Ray sur Amazon EC2](#).

Utilisez un profil d'instance pour autoriser le démon à télécharger des données de trace vers X-Ray. Pour de plus amples informations, veuillez consulter [Autoriser le daemon à envoyer des données à X-Ray](#).

AWS Elastic Beanstalk et AWS X-Ray

AWS Elastic Beanstalk les plateformes incluent le daemon X-Ray. Vous pouvez [exécuter le daemon](#) en définissant une option dans la console Elastic Beanstalk ou à l'aide d'un fichier de configuration.

Sur la plateforme Java SE, vous pouvez utiliser un fichier Buildfile pour créer votre application avec Maven ou Gradle sur l'instance. Le SDK X-Ray pour Java AWS SDK pour Java est disponible auprès de Maven. Vous pouvez donc déployer uniquement le code de votre application et le compiler sur instance afin d'éviter de regrouper et de télécharger toutes vos dépendances.

Vous pouvez utiliser les propriétés de l'environnement Elastic Beanstalk pour configurer le SDK X-Ray. La méthode utilisée par Elastic Beanstalk pour transmettre les propriétés de l'environnement à votre application varie en fonction de la plateforme. Utilisez les variables d'environnement ou les propriétés système du SDK X-Ray en fonction de votre plate-forme.

- [Plateforme Node.js](#) — Utiliser des [variables d'environnement](#)
- [Plateforme Java SE](#) — Utiliser des [variables d'environnement](#)
- [Plateforme Tomcat](#) — Utiliser les propriétés [du système](#)

Pour plus d'informations, consultez [la section Configuration du AWS X-Ray débogage](#) dans le guide du AWS Elastic Beanstalk développeur.

Elastic Load Balancing et AWS X-Ray

Les équilibreurs de charge des applications Elastic Load Balancing ajoutent un identifiant de trace aux requêtes HTTP entrantes dans un en-tête nommé `X-Amzn-Trace-Id`.

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

Format d'identification X-Ray Trace

Un X-Ray `trace_id` est composé de trois chiffres séparés par des tirets. Par exemple, `1-58406520-a006649127e371903a2de979`. Cela consiste notamment à :

- Le numéro de version, qui est `1`.
- L'heure de la demande initiale sous Unix Epoch Time en utilisant 8 chiffres hexadécimaux.

Par exemple, le 1er décembre 2016 à 10 h 00 PST est exprimé en `1480615200` secondes ou `58406520` en chiffres hexadécimaux.

- Identifiant 96 bits unique au monde pour la trace en 24 chiffres hexadécimaux.

Les équilibreurs de charge n'envoient pas de données à X-Ray et n'apparaissent pas sous forme de nœud sur votre carte de service.

Pour plus d'informations, consultez la section [Request Tracing for Your Application Load Balancer](#) dans le manuel Elastic Load Balancing Developer Guide.

Amazon EventBridge et AWS X-Ray

AWS X-Ray s'intègre EventBridge à Amazon pour suivre les événements transmis EventBridge. [Si un service instrumenté avec le SDK X-Ray envoie des événements à EventBridge, le contexte de trace est propagé aux cibles d'événements en aval dans l'en-tête de suivi](#). Le SDK X-Ray récupère

automatiquement l'en-tête de traçage et l'applique à toute instrumentation ultérieure. Cette continuité permet aux utilisateurs de suivre, d'analyser et de déboguer l'ensemble des services en aval et fournit une vue plus complète de leur système.

Pour plus d'informations, voir [EventBridge X-Ray Integration](#) dans le guide de EventBridge l'utilisateur.

Affichage de la source et des cibles sur la carte du service X-Ray

La [carte de suivi](#) X-Ray affiche un nœud d' EventBridge événement qui connecte les services source et cible, comme dans l'exemple suivant :



Propagation du contexte de suivi vers les cibles des événements

Le SDK X-Ray permet à la source d' EventBridge événements de propager le contexte de trace aux cibles d'événements en aval. Les exemples spécifiques au langage suivants illustrent l'appel EventBridge à partir d'une fonction Lambda sur laquelle le suivi [actif](#) est activé :

Java

Ajoutez les dépendances nécessaires pour X-Ray :

- [AWS X-Ray SDK pour Java](#)
- [AWS X-Ray SDK Recorder pour Java](#)

```
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

Ajoutez la dépendance suivante à votre fichier requirements.txt :

```
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#

Ajoutez les packages X-Ray suivants à vos dépendances C# :

```
<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 et AWS X-Ray

Vous pouvez l'utiliser AWS X-Ray pour suivre vos AWS Lambda fonctions. Lambda exécute le [daemon X-Ray](#) et enregistre un segment contenant des détails sur l'invocation et l'exécution de la fonction. Pour une instrumentation plus poussée, vous pouvez associer le SDK X-Ray à votre fonction pour enregistrer les appels sortants et ajouter des annotations et des métadonnées.

Si votre fonction Lambda est appelée par un autre service instrumenté, Lambda suit les requêtes qui ont déjà été échantillonnées sans aucune configuration supplémentaire. Le service en amont peut être une application Web instrumentée ou une autre fonction Lambda. Votre service peut appeler la fonction directement avec un client AWS SDK instrumenté ou en appelant une API API Gateway avec un client HTTP instrumenté.

AWS X-Ray prend en charge le suivi des applications pilotées par des événements à l'aide AWS Lambda d'Amazon SQS. Utilisez la CloudWatch console pour voir une vue connectée de chaque demande mise en file d'attente avec Amazon SQS et traitée par une fonction Lambda en aval. Les traces provenant des producteurs de messages en amont sont automatiquement liées aux traces provenant des nœuds consommateurs Lambda en aval, créant ainsi une end-to-end vue de l'application. Pour plus d'informations, consultez la section [Suivi des applications pilotées par des événements](#).

 Note

Si les traces sont activées pour une fonction Lambda en aval, vous devez également activer les traces pour la fonction Lambda racine qui appelle la fonction en aval afin que la fonction en aval puisse générer des traces.

Si votre fonction Lambda s'exécute selon un calendrier ou est invoquée par un service qui n'est pas instrumenté, vous pouvez configurer Lambda pour échantillonner et enregistrer les appels avec un suivi actif.

Pour configurer l'intégration de X-Ray sur une AWS Lambda fonction

1. Ouvrez la [AWS Lambda console](#).
2. Sélectionnez Fonctions dans la barre de navigation de gauche.
3. Choisissez votre fonction.
4. Dans l'onglet Configuration, faites défiler l'écran vers le bas jusqu'à la carte Outils de surveillance supplémentaires. Vous pouvez également trouver cette carte en sélectionnant Outils de surveillance et d'exploitation dans le volet de navigation de gauche.
5. Tâche de sélection Modifier.
6. Sous AWS X-Ray, activez le Suivi actif.

Lors des exécutions avec un SDK X-Ray correspondant, Lambda exécute également le démon X-Ray.

X-Ray SDKs sur Lambda

- X-Ray SDK pour Go — Go 1.7 et environnements d'exécution plus récents
- Kit de développement X-Ray pour Java — environnement d'exécution Java 8
- Kit de développement X-Ray pour Node.js — Node.js 4.3 et versions ultérieures
- Kit de développement X-Ray pour Python — Python 2.7, Python 3.6 et environnements d'exécution plus récents
- Kit de développement X-Ray pour .NET — .NET Core 2.0 et environnements d'exécution plus récents

Pour utiliser le SDK X-Ray sur Lambda, associez-le à votre code de fonction chaque fois que vous créez une nouvelle version. Vous pouvez instrumenter vos fonctions Lambda avec les mêmes méthodes que celles que vous utilisez pour instrumenter les applications exécutées sur d'autres services. La principale différence est que vous n'utilisez pas le kit de développement logiciel pour instrumenter les demandes entrantes, prendre des décisions d'échantillonnage et créer des segments.

L'autre différence entre l'instrumentation des fonctions Lambda et des applications Web est que le segment que Lambda crée et envoie à X-Ray ne peut pas être modifié par le code de votre fonction. Vous pouvez créer des sous-segments et enregistrer des annotations et des métadonnées sur eux, mais vous ne pouvez pas ajouter d'annotations et de métadonnées au segment parent.

Pour plus d'informations, consultez la section [Utilisation de AWS X-Ray](#) dans le manuel du AWS Lambda développeur.

Amazon SNS et AWS X-Ray

[Vous pouvez utiliser AWS X-Ray Amazon Simple Notification Service \(Amazon SNS\) pour suivre et analyser les demandes au fur et à mesure qu'elles transitent par vos rubriques SNS vers vos services d'abonnement pris en charge par le SNS.](#) Utilisez le suivi X-Ray avec Amazon SNS pour analyser les temps de latence de vos messages et de leurs services principaux, par exemple le temps passé par une demande dans un sujet et le temps nécessaire pour transmettre le message à chacun des abonnements du sujet. Amazon SNS prend en charge le suivi X-Ray pour les rubriques standard et FIFO.

Si vous publiez sur une rubrique Amazon SNS à partir d'un service déjà équipé de X-Ray, Amazon SNS transmet le contexte de suivi de l'éditeur aux abonnés. En outre, vous pouvez activer le suivi actif pour envoyer à X-Ray des données de segment relatives à vos abonnements Amazon SNS pour les messages publiés à partir d'un client SNS instrumenté. [Activez le suivi actif](#) pour une rubrique Amazon SNS à l'aide de la console Amazon SNS ou à l'aide de l'API ou de la CLI Amazon SNS. Consultez [Instrumentation de votre application](#) pour plus d'informations sur l'instrumentation de vos clients SNS.

Configurer le suivi actif Amazon SNS

Vous pouvez utiliser la console Amazon SNS, la AWS CLI ou le SDK pour configurer le suivi actif Amazon SNS.

Lorsque vous utilisez la console Amazon SNS, Amazon SNS tente de créer les autorisations nécessaires pour que SNS appelle X-Ray. La tentative peut être rejetée si vous ne disposez pas des autorisations suffisantes pour modifier les politiques relatives aux ressources de X-Ray. Pour plus d'informations sur ces autorisations, consultez la section [Gestion des identités et des accès dans Amazon SNS et Exemples de cas de contrôle d'accès Amazon SNS](#) dans le guide du développeur Amazon Simple Notification Service. Pour plus d'informations sur l'activation du suivi actif à l'aide de la console Amazon SNS, consultez la section [Activation du suivi actif sur une rubrique Amazon SNS](#) du manuel Amazon Simple Notification Service Developer Guide.

Lorsque vous utilisez la AWS CLI ou le SDK pour activer le suivi actif, vous devez configurer manuellement les autorisations à l'aide de politiques basées sur les ressources. [PutResourcePolicy](#) À utiliser pour configurer X-Ray avec la politique basée sur les ressources nécessaire pour permettre à Amazon SNS d'envoyer des traces à X-Ray.

Exemple Exemple de politique basée sur les ressources X-Ray pour le suivi actif d'Amazon SNS

Cet exemple de document de politique précise les autorisations dont Amazon SNS a besoin pour envoyer des données de suivi à X-Ray :

```
{
  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"
        }
    }
}

```

Utilisez la CLI pour créer une politique basée sur les ressources qui autorise Amazon SNS à envoyer des données de suivi à X-Ray :

```

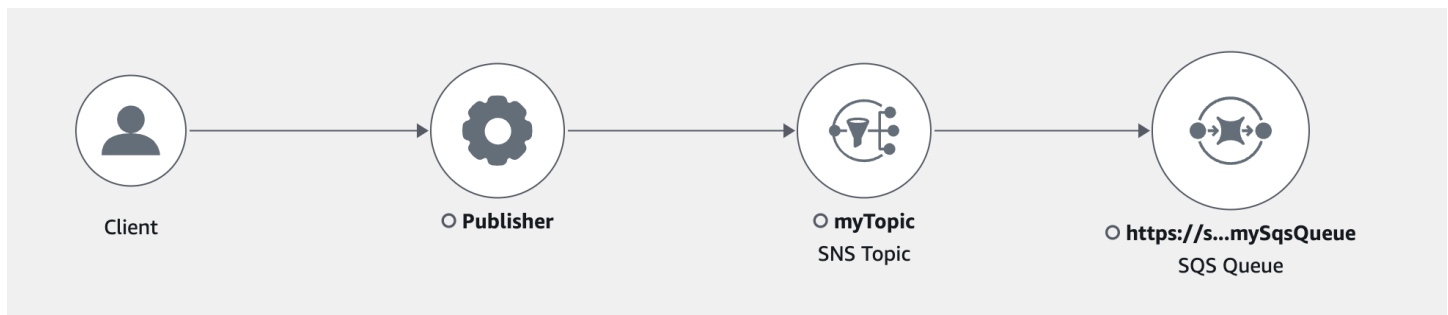
aws xray put-resource-policy --policy-name MyResourcePolicy --policy-document
'{ "Version": "2012-10-17", "Statement": [ { "Sid": "SNSAccess", "Effect": "Allow",
"Principal": { "Service": "sns.amazonaws.com" }, "Action": [ "xray:PutTraceSegments",
"xray:GetSamplingRules", "xray:GetSamplingTargets" ],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

```

Pour utiliser ces exemples, remplacez *partition*, *region*, *account-id*, et *topic-name* par votre AWS partition, région, ID de compte et nom de rubrique Amazon SNS spécifiques. Pour autoriser toutes les rubriques Amazon SNS à envoyer des données de suivi à X-Ray, remplacez le nom de la rubrique par. *

Afficher les traces des éditeurs et des abonnés Amazon SNS dans la console X-Ray

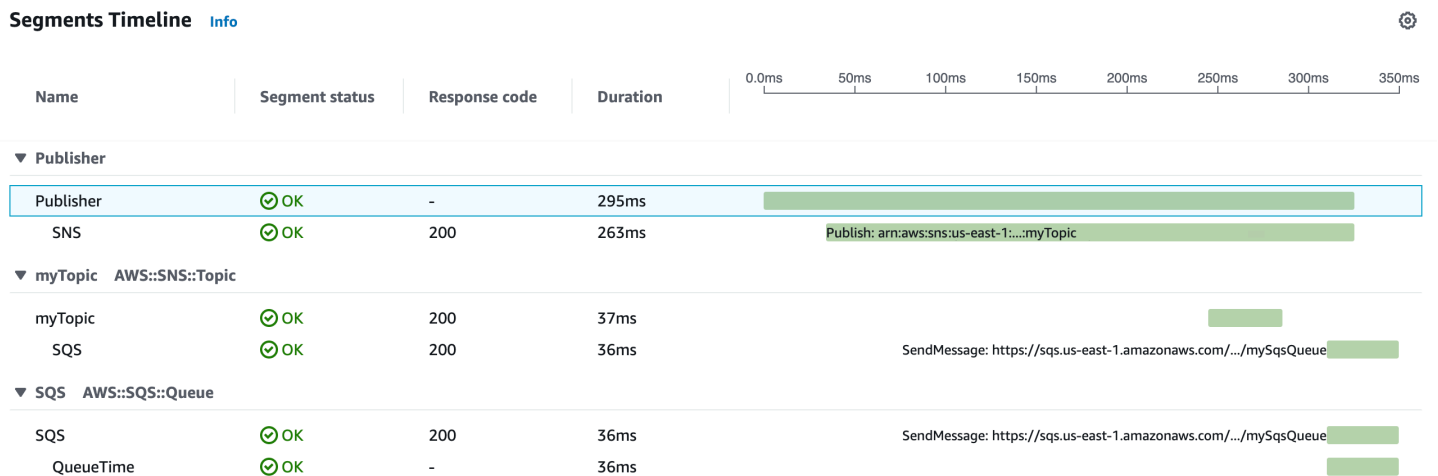
Utilisez la console X-Ray pour consulter une carte de trace et les détails des traces qui affichent une vue connectée des éditeurs et des abonnés Amazon SNS. Lorsque le suivi actif d'Amazon SNS est activé pour une rubrique, la carte de suivi et la carte des détails des traces X-Ray affichent les nœuds connectés pour les éditeurs Amazon SNS, la rubrique Amazon SNS et les abonnés en aval :



Après avoir sélectionné un suivi couvrant un éditeur et un abonné Amazon SNS, la page des détails du suivi X-Ray affiche une carte des détails du suivi et une chronologie des segments.

Exemple de chronologie avec l'éditeur et l'abonné Amazon SNS

Cet exemple montre une chronologie qui inclut un éditeur Amazon SNS qui envoie un message à une rubrique Amazon SNS, qui est traité par un abonné Amazon SQS.



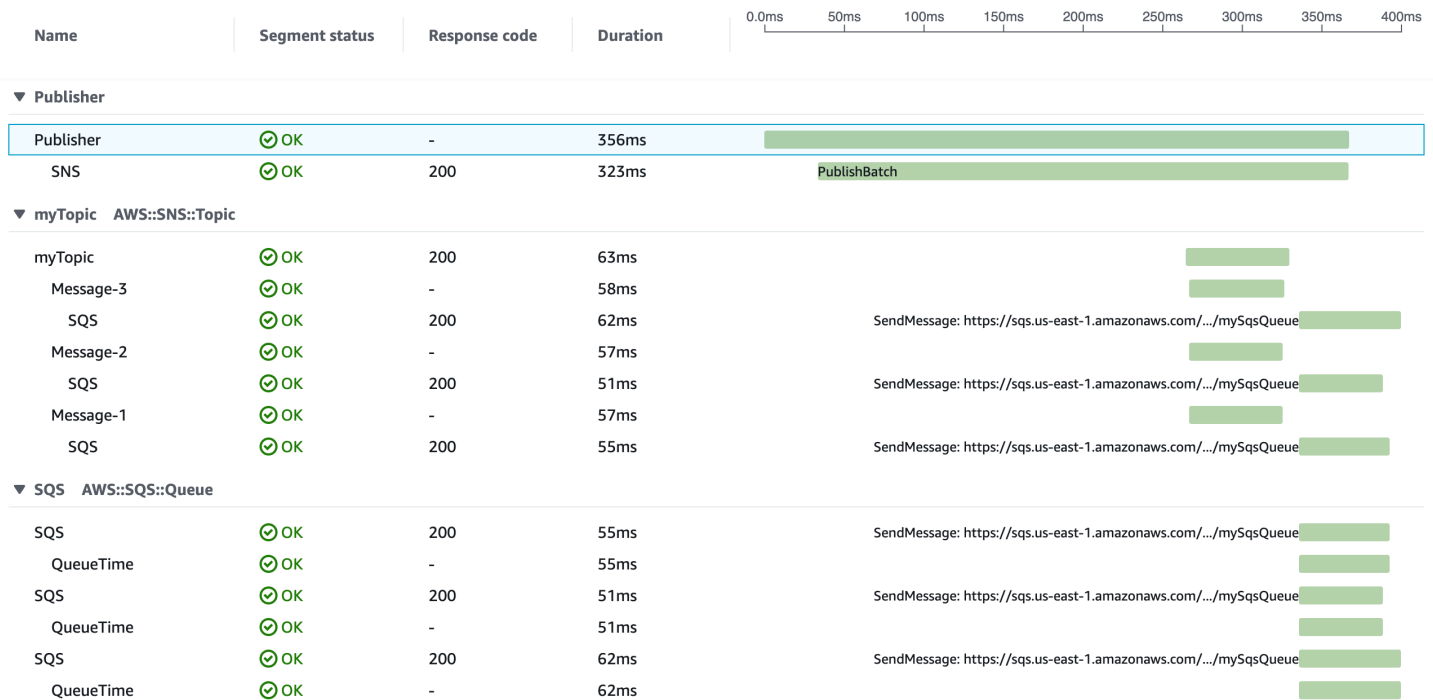
L'exemple de chronologie ci-dessus fournit des informations sur le flux de messages Amazon SNS :

- Le segment SNS représente la durée aller-retour de l'appel d'PublishAPI du client.
- Le segment MyTopic représente la latence de la réponse Amazon SNS à la demande de publication.
- Le sous-segment SQS représente le temps aller-retour nécessaire à Amazon SNS pour publier le message dans une file d'attente Amazon SQS.
- Le temps entre le segment MyTopic et le sous-segment SQS représente le temps que le message passe dans le système Amazon SNS.

Exemple Exemple de chronologie avec des messages Amazon SNS groupés

Si plusieurs messages Amazon SNS sont regroupés par lots au sein d'une même trace, la chronologie des segments affiche les segments qui représentent chaque message traité.

Segments Timeline [Info](#)



AWS Step Functions et AWS X-Ray

AWS X-Ray s'intègre AWS Step Functions à Step Functions pour suivre et analyser les demandes de Step Functions. Vous pouvez visualiser les composants de votre machine d'état, identifier les goulots d'étranglement liés aux performances et résoudre les demandes qui ont entraîné une erreur. Pour plus d'informations, reportez-vous AWS X-Ray à [Step Functions](#) dans le manuel du AWS Step Functions développeur.

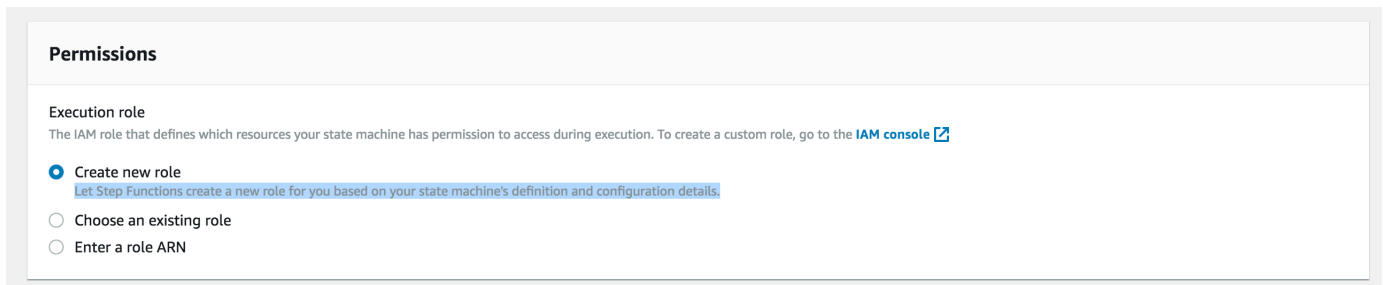
Pour activer le traçage X-Ray lors de la création d'une nouvelle machine à états

1. Ouvrez la console Step Functions à l'adresse <https://console.aws.amazon.com/states/>.
2. Choisissez Create a state machine.
3. Sur la page Définir la machine à états, sélectionnez Auteur avec des extraits de code ou Démarrer avec un modèle. Si vous choisissez d'exécuter un exemple de projet, vous ne pouvez pas activer le suivi X-Ray lors de sa création. Activez plutôt le suivi X-Ray après avoir créé votre machine à états.

4. Choisissez Suivant.
5. Sur la page Spécifier les détails, configurez votre machine d'état.
6. Choisissez Enable X-Ray Tracing.

Pour activer le traçage par rayons X dans une machine à états existante

1. Dans la console Step Functions, sélectionnez la machine à états pour laquelle vous souhaitez activer le suivi.
2. Choisissez Modifier.
3. Choisissez Enable X-Ray Tracing.
4. (Facultatif) Générez automatiquement un nouveau rôle pour votre machine d'état afin d'inclure les autorisations X-Ray en choisissant Create new role dans la fenêtre Permissions.



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. Choisissez Save (Enregistrer).

Note

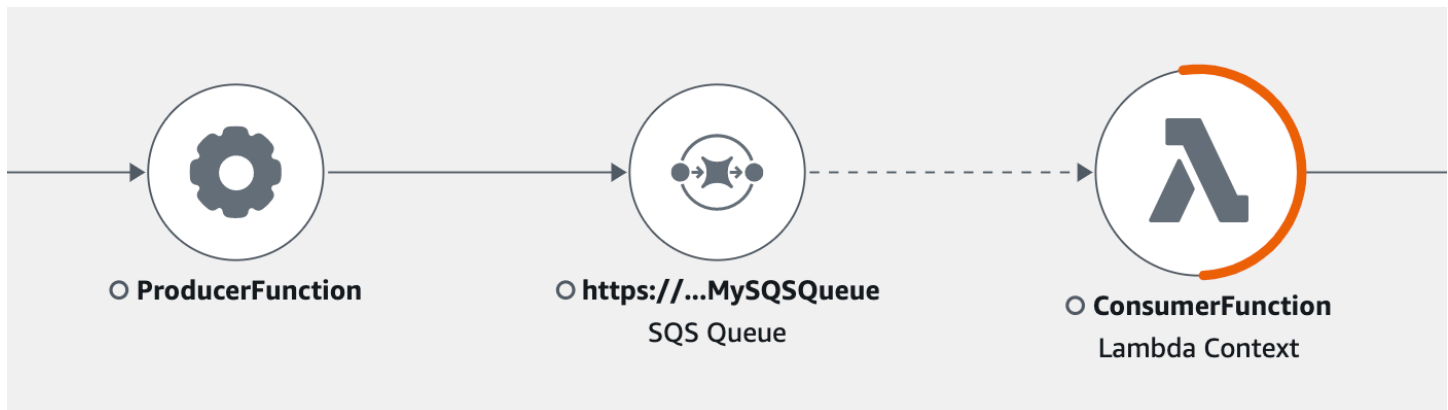
Lorsque vous créez une nouvelle machine à états, elle est automatiquement tracée si la demande est échantillonnée et si le suivi est activé dans un service en amont tel qu'Amazon API Gateway ou AWS Lambda. Pour toute machine à états existante non configurée via la console, par exemple via un AWS CloudFormation modèle, vérifiez que vous disposez d'une politique IAM qui accorde des autorisations suffisantes pour activer les traces X-Ray.

Amazon SQS et AWS X-Ray

AWS X-Ray s'intègre à Amazon Simple Queue Service (Amazon SQS) pour suivre les messages transmis via une file d'attente Amazon SQS. Si un service assure le suivi des demandes à l'aide du SDK X-Ray, Amazon SQS peut envoyer l'en-tête de suivi et continuer à propager le suivi d'origine de

l'expéditeur au consommateur avec un identifiant de suivi cohérent. La continuité du suivi permet aux utilisateurs de suivre, d'analyser et de déboguer les services en aval d'un bout à l'autre.

AWS X-Ray prend en charge le suivi des applications pilotées par des événements à l'aide d'Amazon SQS et. AWS Lambda Utilisez la CloudWatch console pour voir une vue connectée de chaque demande mise en file d'attente avec Amazon SQS et traitée par une fonction Lambda en aval. Les traces provenant des producteurs de messages en amont sont automatiquement liées aux traces provenant des nœuds consommateurs Lambda en aval, créant ainsi une end-to-end vue de l'application. Pour plus d'informations, consultez la section [Suivi des applications pilotées par des événements](#).



Amazon SQS prend en charge l'instrumentation d'en-tête de suivi suivante :

- **En-tête HTTP par défaut** — Le SDK X-Ray remplit automatiquement l'en-tête de trace en tant qu'en-tête HTTP lorsque vous appelez Amazon SQS via le SDK. AWS L'en-tête de suivi par défaut est transporté par `X-Amzn-Trace-Id` et correspond à tous les messages inclus dans une demande [SendMessage](#) ou [SendMessageBatch](#). Pour en savoir plus sur l'en-tête HTTP par défaut, consultez [En-tête de suivi](#).
- **AWSTraceHeader**Attribut système — `AWSTraceHeader` Il s'agit d'un [attribut de système de messagerie](#) réservé par Amazon SQS pour contenir l'en-tête X-Ray Trace avec les messages dans la file d'attente. `AWSTraceHeader` peut être utilisé même lorsque l'instrumentation automatique via le SDK X-Ray ne l'est pas, par exemple lors de la création d'un SDK de traçage pour une nouvelle langue. Lorsque les deux instrumentations d'en-tête sont définies, l'attribut système de message remplace l'en-tête de suivi HTTP.

Lorsqu'il est exécuté sur Amazon EC2, Amazon SQS prend en charge le traitement d'un message à la fois. Cela s'applique lors de l'exécution sur un hôte sur site et lors de l'utilisation de services de conteneur AWS Fargate, tels qu'Amazon ECS ou AWS App Mesh.

L'en-tête de trace est exclu à la fois de la taille des messages Amazon SQS et des quotas d'attributs des messages. L'activation du suivi X-Ray ne dépassera pas vos quotas Amazon SQS. Pour en savoir plus sur les AWS quotas, consultez [Amazon SQS Quotas](#).

Envoi de l'en-tête de suivi HTTP

Les composants d'expéditeur d'Amazon SQS peuvent envoyer automatiquement l'en-tête de trace via l'appel [SendMessageBatch](#) ou [SendMessage](#). Lorsque les clients du AWS SDK sont instrumentés, ils peuvent être automatiquement suivis dans toutes les langues prises en charge par le SDK X-Ray. Les ressources tracées Services AWS et auxquelles vous accédez au sein de ces services (par exemple, un compartiment Amazon S3 ou une file d'attente Amazon SQS) apparaissent sous forme de nœuds en aval sur la carte de suivi de la console X-Ray.

Pour savoir comment suivre les appels du AWS SDK dans la langue de votre choix, consultez les rubriques suivantes dans les rubriques prises en charge SDKs :

- Go – [Suivi des appels du AWS SDK avec le SDK X-Ray pour Go](#)
- Java — [Suivi des appels du AWS SDK avec le SDK X-Ray pour Java](#)
- Node.js – [Suivi des appels du AWS SDK avec le SDK X-Ray pour Node.js](#)
- Python – [Suivi des appels du AWS SDK avec le SDK X-Ray pour Python](#)
- Ruby – [Suivi des appels du AWS SDK avec le SDK X-Ray pour Ruby](#)
- .NET – [Suivi des appels du AWS SDK avec le SDK X-Ray pour .NET](#)

Récupération de l'en-tête de suivi et récupération du contexte de suivi

Si vous utilisez un consommateur Lambda en aval, la propagation du contexte de trace est automatique. Pour poursuivre la propagation du contexte avec d'autres clients Amazon SQS, vous devez instrumenter manuellement le transfert vers le composant récepteur.

La récupération du contexte de suivi comporte trois étapes principales :

- Recevoir le message de la file d'attente pour l'attribut `AWSTraceHeader` en appelant l'API [ReceiveMessage](#).
- Récupérer l'en-tête de suivi à partir de l'attribut.
- Récupérer l'ID de suivi à partir de l'en-tête. Vous pouvez également ajouter d'autres métriques au segment.

Voici un exemple d'implémentation écrit avec le SDK X-Ray pour Java.

Exemple : Récupération de l'en-tête de suivi et récupération du contexte de suivi

```
// Receive the message from the queue, specifying the "AWSTraceHeader"
ReceiveMessageRequest receiveMessageRequest = new ReceiveMessageRequest()
    .withQueueUrl(QUEUE_URL)
    .withAttributeNames("AWSTraceHeader");
List<Message> messages = sqs.receiveMessage(receiveMessageRequest).getMessages();

if (!messages.isEmpty()) {
    Message message = messages.get(0);

    // Retrieve the trace header from the AWSTraceHeader message system attribute
    String traceHeaderStr = message.getAttributes().get("AWSTraceHeader");
    if (traceHeaderStr != null) {
        TraceHeader traceHeader = TraceHeader.fromString(traceHeaderStr);

        // Recover the trace context from the trace header
        Segment segment = AWSXRay.getCurrentSegment();
        segment.setTraceId(traceHeader.getRootTraceId());
        segment.setParentId(traceHeader.getParentId());

        segment.setSampled(traceHeader.getSampled().equals(TraceHeader.SampleDecision.SAMPLED));
    }
}
```

Amazon S3 et AWS X-Ray

AWS X-Ray s'intègre à Amazon S3 pour suivre les demandes en amont visant à mettre à jour les compartiments S3 de votre application. Si un service effectue le suivi des demandes à l'aide du SDK X-Ray, Amazon S3 peut envoyer les en-têtes de suivi aux abonnés aux événements en aval AWS Lambda, tels qu'Amazon SQS et Amazon SNS. X-Ray active les messages de suivi pour les notifications d'événements Amazon S3.

Vous pouvez utiliser la carte de suivi X-Ray pour afficher les connexions entre Amazon S3 et les autres services que votre application utilise. Vous pouvez également utiliser la console pour afficher des mesures comme la latence moyenne et les taux de défaillance. Pour plus d'informations sur la console X-Ray, consultez [Utiliser la console X-Ray](#).

Amazon S3 prend en charge l'instrumentation d'en-tête HTTP par défaut. Le SDK X-Ray remplit automatiquement l'en-tête de trace en tant qu'en-tête HTTP lorsque vous appelez Amazon S3 via le AWS SDK. L'en-tête de trace par défaut est porté par `X-Amzn-Trace-Id`. Pour en savoir plus sur le suivi des en-têtes, consultez la [En-tête de suivi](#) page conceptuelle. La propagation du contexte de trace Amazon S3 prend en charge les abonnés suivants : Lambda, SQS et SNS. Étant donné que SQS et SNS n'émettent pas eux-mêmes de données de segment, elles n'apparaîtront pas dans votre trace ou votre carte de trace lorsqu'elles sont déclenchées par S3, même s'ils propagent l'en-tête de suivi aux services en aval.

Configuration des notifications d'événements Amazon S3

Grâce à la fonctionnalité de notification Amazon S3, vous recevez des notifications lorsque certains événements se produisent dans votre compartiment. Ces notifications peuvent ensuite être propagées aux destinations suivantes au sein de votre application :

- Amazon Simple Notification Service (Amazon SNS)
- Amazon Simple Queue Service (Amazon SQS)
- AWS Lambda

Pour obtenir la liste des événements pris en charge, consultez la section [Types d'événements pris en charge dans le guide du développeur Amazon S3](#).

Amazon SNS et Amazon SQS

Pour publier des notifications sur une rubrique SNS ou une file d'attente SQS, vous devez d'abord accorder des autorisations Amazon S3. Pour accorder ces autorisations, vous devez associer une politique AWS Identity and Access Management (IAM) à la rubrique SNS ou à la file d'attente SQS de destination. Pour en savoir plus sur les politiques IAM requises, consultez la section [Octroi d'autorisations pour publier des messages sur une rubrique SNS ou une file d'attente SQS](#).

Pour plus d'informations sur l'intégration de SNS et SQS à X-Ray, [Amazon SNS et AWS X-Ray](#) voir et. [Amazon SQS et AWS X-Ray](#)

AWS Lambda

Lorsque vous utilisez la console Amazon S3 pour configurer les notifications d'événements sur un compartiment S3 pour une fonction Lambda, la console définit les autorisations nécessaires sur la fonction Lambda afin qu'Amazon S3 soit autorisé à appeler la fonction depuis le compartiment. Pour

plus d'informations, consultez [Comment activer et configurer les notifications d'événements pour un compartiment S3 ?](#) dans le guide de l'utilisateur de la console Amazon Simple Storage Service.

Vous pouvez également accorder à Amazon S3 des autorisations AWS Lambda pour appeler votre fonction Lambda. Pour plus d'informations, consultez [Tutoriel : Utilisation de AWS Lambda avec Amazon S3](#) dans le Guide du développeur AWS Lambda.

Pour plus d'informations sur l'intégration de Lambda à X-Ray, consultez [Instrumentation du code Java dans Lambda](#). AWS

Instrumentation de votre application pour AWS X-Ray

L'instrumentation de votre application implique l'envoi de données de suivi pour les demandes entrantes et sortantes et d'autres événements au sein de votre application, ainsi que des métadonnées relatives à chaque demande. Il existe plusieurs options d'instrumentation différentes que vous pouvez choisir ou combiner, en fonction de vos besoins particuliers :

- Instrumentation automatique : instrumentez votre application sans aucune modification de code, généralement via des modifications de configuration, l'ajout d'un agent d'instrumentation automatique ou d'autres mécanismes.
- Instrumentation de bibliothèque : apportez des modifications minimales au code de l'application pour ajouter une instrumentation prédéfinie ciblant des bibliothèques ou des frameworks spécifiques, tels que le AWS SDK, les clients HTTP Apache ou les clients SQL.
- Instrumentation manuelle : ajoutez un code d'instrumentation à votre application à chaque endroit où vous souhaitez envoyer des informations de suivi.

Il existe plusieurs SDKs agents et outils qui peuvent être utilisés pour instrumenter votre application pour le traçage par rayons X.

Rubriques

- [Instrumentation de votre application avec le AWS Distro pour OpenTelemetry](#)
- [Instrumentation de votre application avec AWS X-Ray SDKs](#)
- [Choisir entre The AWS Distro for OpenTelemetry et X-Ray SDKs](#)

Instrumentation de votre application avec le AWS Distro pour OpenTelemetry

The AWS Distro for OpenTelemetry (ADOT) est une AWS distribution basée sur le projet Cloud Native Computing Foundation (CNCF). OpenTelemetry fournit un ensemble unique d'open source APIs, de bibliothèques et d'agents pour collecter des traces et des métriques distribuées. Cette boîte à outils est une distribution de OpenTelemetry composants en amont SDKs, notamment des agents d'auto-instrumentation et des collecteurs testés, optimisés, sécurisés et pris en charge par AWS.

Avec ADOT, les ingénieurs peuvent instrumenter leurs applications une seule fois et envoyer des métriques et des traces corrélées à plusieurs solutions de AWS surveillance CloudWatch AWS X-Ray, notamment Amazon et Amazon OpenSearch Service.

L'utilisation de X-Ray avec ADOT nécessite deux composants : un OpenTelemetry SDK activé pour une utilisation avec X-Ray, et le AWS Distro for OpenTelemetry Collector activé pour une utilisation avec X-Ray. Pour plus d'informations sur l'utilisation de la AWS distribution pour OpenTelemetry with AWS X-Ray et d'autres Services AWS, consultez la [AWS distribution pour OpenTelemetry](#) la documentation.

Pour plus d'informations sur la prise en charge et l'utilisation des langues, consultez [AWS Observability on GitHub](#).

Note

Vous pouvez désormais utiliser l' CloudWatch agent pour collecter des métriques, des journaux et des traces à partir d' EC2 instances Amazon et de serveurs sur site. CloudWatch les versions 1.300025.0 et ultérieures de l'agent peuvent collecter des traces depuis notre client [OpenTelemetryX-Ray](#) et SDKs les envoyer à X-Ray. L'utilisation de l' CloudWatch agent au lieu du daemon AWS Distro for OpenTelemetry (ADOT) Collector ou X-Ray pour collecter des traces peut vous aider à réduire le nombre d'agents que vous gérez. Consultez la rubrique [CloudWatch relative aux agents](#) dans le guide de CloudWatch l'utilisateur pour plus d'informations.

ADOT inclut les éléments suivants :

- [AWS Distro pour Go OpenTelemetry](#)
- [AWS Distro pour Java OpenTelemetry](#)
- [AWS Distro pour OpenTelemetry JavaScript](#)
- [AWS Distro pour Python OpenTelemetry](#)
- [AWS Distro pour .NET OpenTelemetry](#)

[ADOT inclut actuellement la prise en charge de l'instrumentation automatique pour Java et Python. En outre, ADOT permet l'instrumentation automatique des fonctions AWS Lambda et de leurs requêtes en aval à l'aide des environnements d'exécution Java, Node.js et Python, via des couches Lambda gérées par ADOT.](#)

ADOT SDKs pour Java et Go prennent en charge les règles d'échantillonnage centralisées X-Ray. Si vous avez besoin d'assistance pour les règles d'échantillonnage X-Ray dans d'autres langues, pensez à utiliser un AWS X-Ray SDK.

Note

Vous pouvez maintenant envoyer la trace du W3C IDs à X-Ray. Par défaut, les traces créées avec OpenTelemetry ont un format d'ID de trace basé sur la [spécification du W3C Trace Context](#). Ce format est différent du format de trace IDs créé à l'aide d'un SDK X-Ray ou par AWS des services intégrés à X-Ray. Pour vous assurer que les traces IDs au format W3C sont acceptées par X-Ray, vous devez utiliser la version 0.86.0 ou ultérieure d'[AWS X-Ray Exporter](#), qui est incluse dans [ADOT Collector](#) version 0.34.0 et ultérieure. Les versions précédentes de l'exportateur valident les horodatages des identifiants de trace, ce qui peut entraîner le rejet du suivi IDs du W3C.

Instrumentation de votre application avec AWS X-Ray SDKs

AWS X-Ray inclut un ensemble de langages spécifiques SDKs pour instrumenter votre application afin d'envoyer des traces à X-Ray. Chaque kit de développement X-Ray fournit les éléments suivants :

- Des intercepteurs à ajouter à votre code pour suivre les demandes HTTP entrantes
- Des gestionnaires de clients pour instrumenter les clients du AWS SDK que votre application utilise pour appeler d'autres Services AWS
- Un client HTTP pour instrumenter les appels vers d'autres services Web HTTP internes et externes

X-Ray prend SDKs également en charge l'instrumentation des appels aux bases de données SQL, l'instrumentation client automatique du AWS SDK et d'autres fonctionnalités. Au lieu d'envoyer les données de suivi directement à X-Ray, le SDK envoie des documents de segments JSON à un processus démon qui écoute le trafic UDP. Le [daemon X-Ray](#) met en mémoire tampon les segments d'une file d'attente et les télécharge dans X-Ray par lots.

Les informations spécifiques à la langue suivantes SDKs sont fournies :

- [AWS X-Ray SDK pour Go](#)
- [AWS X-Ray SDK pour Java](#)

- [AWS X-Ray SDK pour Node.js](#)
- [AWS X-Ray SDK pour Python](#)
- [AWS X-Ray SDK pour .NET](#)
- [AWS X-Ray SDK pour Ruby](#)

X-Ray inclut actuellement la prise en charge de l'instrumentation automatique pour [Java](#).

Choisir entre The AWS Distro for OpenTelemetry et X-Ray SDKs

Les appareils SDKs inclus avec X-Ray font partie d'une solution d'instrumentation étroitement intégrée proposée par AWS. The AWS Distro for OpenTelemetry fait partie d'une solution industrielle plus large dans laquelle X-Ray n'est qu'une des nombreuses solutions de traçage. Vous pouvez implémenter le end-to-end traçage dans X-Ray en utilisant l'une ou l'autre approche, mais il est important de comprendre les différences afin de déterminer l'approche la plus utile pour vous.

Nous vous recommandons d'instrumenter votre application avec le AWS Distro OpenTelemetry si vous avez besoin des éléments suivants :

- La possibilité d'envoyer des traces à plusieurs backends de suivi différents sans avoir à réinstrumenter votre code
- Support pour un grand nombre d'instrumentations de bibliothèque pour chaque langue, maintenues par la communauté OpenTelemetry
- Des couches Lambda entièrement gérées qui regroupent tout ce dont vous avez besoin pour collecter des données de télémétrie, sans qu'il soit nécessaire de modifier le code lors de l'utilisation de Java, Python ou Node.js

Note

AWS Distro for OpenTelemetry offre une expérience de démarrage plus simple pour l'instrumentation de vos fonctions Lambda. Cependant, en raison OpenTelemetry des offres de flexibilité, votre fonction Lambda nécessitera de la mémoire supplémentaire et les appels peuvent connaître une augmentation de la latence de démarrage à froid, ce qui peut entraîner des frais supplémentaires. Si vous optez pour une faible latence et que vous n'avez pas besoin OpenTelemetry de fonctionnalités avancées telles que des destinations dorsales configurables dynamiquement, vous pouvez utiliser le SDK AWS X-Ray pour instrumenter votre application.

Nous vous recommandons de choisir un SDK X-Ray pour instrumenter votre application si vous avez besoin des éléments suivants :

- Une solution à fournisseur unique étroitement intégrée
- Intégration aux règles d'échantillonnage centralisées de X-Ray, notamment la possibilité de configurer les règles d'échantillonnage depuis la console X-Ray et de les utiliser automatiquement sur plusieurs hôtes, lors de l'utilisation de Node.js, Python, Ruby ou .NET

Recherche de transactions

Transaction Search est une expérience d'analyse interactive que vous pouvez utiliser pour obtenir une visibilité complète de la durée des transactions de votre application. Les spans sont les unités opérationnelles fondamentales d'un traçage distribué et représentent des actions ou des tâches spécifiques dans une application ou un système. Chaque intervalle enregistre les détails d'un segment particulier de la transaction. Ces détails incluent les heures de début et de fin, la durée et les métadonnées associées, qui peuvent inclure des attributs commerciaux tels que le client IDs et la commande IDs. Les travées sont organisées selon une hiérarchie parent-enfant. Cette hiérarchie forme une trace complète cartographiant le flux d'une transaction entre différents composants ou services.

Pour plus d'informations, consultez la section [Recherche de transactions](#).

OpenTelemetry Point de terminaison du protocole (OTLP)

OpenTelemetry est un framework d'observabilité open source qui fournit aux équipes informatiques des protocoles et des outils standardisés pour la collecte et le routage des données de télémétrie. Il fournit un format unifié pour l'instrumentation, la génération, la collecte et l'exportation des données de télémétrie des applications, telles que les métriques, les journaux et les traces, vers des plateformes de surveillance à des fins d'analyse et d'analyse. En les utilisant OpenTelemetry, les équipes peuvent éviter la dépendance vis-à-vis des fournisseurs, garantissant ainsi la flexibilité de leurs solutions d'observabilité.

Vous pouvez l'utiliser OpenTelemetry pour envoyer directement des traces à un point de terminaison OpenTelemetry du protocole (OTLP) et bénéficier des expériences de surveillance des performances des applications out-of-the Box dans [CloudWatch Application Signals](#).

Pour de plus amples informations, veuillez consulter [OpenTelemetry](#).

Utilisation de Go

Il existe deux manières d'instrumenter votre application Go pour envoyer des traces à X-Ray :

- [AWS Distro for OpenTelemetry Go](#) : AWS distribution qui fournit un ensemble de bibliothèques open source permettant d'envoyer des métriques et des traces corrélées à plusieurs solutions de AWS surveillance CloudWatch, notamment Amazon et Amazon OpenSearch Service AWS X-Ray, via [AWS Distro](#) for Collector. OpenTelemetry
- [AWS X-Ray SDK for Go](#) : ensemble de bibliothèques permettant de générer et d'envoyer des traces à X-Ray via le daemon [X-Ray](#).

Pour de plus amples informations, veuillez consulter [Choisir entre The AWS Distro for OpenTelemetry et X-Ray SDKs](#).

AWS Distro pour Go OpenTelemetry

Avec AWS Distro for OpenTelemetry Go, vous pouvez instrumenter vos applications une seule fois et envoyer des métriques et des traces corrélées à plusieurs solutions de AWS surveillance CloudWatch AWS X-Ray, notamment Amazon et Amazon OpenSearch Service. L'utilisation de X-Ray avec AWS Distro pour OpenTelemetry nécessite deux composants : un OpenTelemetry SDK activé pour une utilisation avec X-Ray, et un AWS Distro for OpenTelemetry Collector activé pour une utilisation avec X-Ray.

Pour commencer, consultez la [documentation de AWS Distro for OpenTelemetry Go](#).

Pour plus d'informations sur l'utilisation de la AWS distribution pour OpenTelemetry with AWS X-Ray et d'autres Services AWS, consultez la section [AWS Distro for OpenTelemetry](#) ou la distribution pour la [AWS documentation](#). OpenTelemetry

Pour plus d'informations sur la prise en charge et l'utilisation des langues, consultez [AWS Observability on GitHub](#).

AWS X-Ray SDK pour Go

Le SDK X-Ray pour Go est un ensemble de bibliothèques pour les applications Go qui fournissent des classes et des méthodes permettant de générer et d'envoyer des données de trace au daemon X-Ray. Les données de suivi incluent des informations sur les requêtes HTTP entrantes traitées

par l'application et les appels que l'application envoie aux services en aval à l'aide du AWS SDK, des clients HTTP ou d'un connecteur de base de données SQL. Vous pouvez également créer des segments manuellement et ajouter des informations de débogage dans les annotations et les métadonnées.

Téléchargez le SDK depuis son [GitHub référentiel](#) avec `go get` :

```
$ go get -u github.com/aws/aws-xray-sdk-go/...
```

Pour les applications web, commencez par [utiliser la `xray.Handler` fonction](#) pour suivre les demandes entrantes. Le gestionnaire de messages crée un [segment](#) pour chaque demande suivie et met fin au segment lorsque la réponse est envoyée. Tandis que le segment soit ouvert, vous pouvez utiliser les méthodes du client du SDK pour ajouter des informations au segment et créer des sous-segments afin de suivre les appels en amont. Le kit de développement logiciel (SDK) enregistre aussi automatiquement les exceptions que votre application renvoie quand le segment est ouvert.

Pour les fonctions Lambda appelées par une application ou un service instrumenté, Lambda lit l'[en-tête de suivi et trace automatiquement les requêtes échantillonnées](#). Pour les autres fonctions, vous pouvez [configurer Lambda](#) pour échantillonner et suivre les demandes entrantes. Dans les deux cas, Lambda crée le segment et le fournit au SDK X-Ray.

Note

Sur Lambda, le SDK X-Ray est facultatif. Si vous ne l'utilisez pas dans votre fonction, votre carte de service inclura toujours un nœud pour le service Lambda et un nœud pour chaque fonction Lambda. En ajoutant le SDK, vous pouvez instrumenter votre code de fonction pour ajouter des sous-segments au segment de fonction enregistré par Lambda. Pour plus d'informations, consultez [AWS Lambda et AWS X-Ray](#).

Ensuite, [enveloppez votre client avec un appel à la fonction AWS](#). Cette étape garantit que X-Ray instrumente les appels à toutes les méthodes client. Vous pouvez également [instrumenter les appels vers les bases de données SQL](#).

Après avoir commencé à utiliser le SDK, personnalisez son comportement en [configurant l'enregistreur et le middleware](#). Vous pouvez ajouter des plug-ins pour enregistrer les données sur les ressources de calcul exécutant votre application, personnaliser le comportement d'échantillonnage en définissant des règles d'échantillonnage et définir le niveau de journal afin d'afficher plus ou moins d'informations du kit de développement logiciel dans vos journaux d'application.

Enregistrez les informations supplémentaires sur les demandes et le travail que votre application effectue dans les [annotations et les métadonnées](#). Les annotations sont de simples paires clé-valeur, indexées en vue de leur utilisation avec les [expressions de filtre](#), de telle sorte que vous pouvez explorer les suivis qui contiennent des données spécifiques. Les entrées des métadonnées sont moins restrictives et peuvent enregistrer des objets et tableaux entiers (tout ce qui peut être sérialisé en JSON).

Annotations et métadonnées

Les annotations et les métadonnées sont du texte arbitraire que vous ajoutez aux segments avec le kit de développement X-Ray. Les annotations sont indexées pour être utilisées avec les expressions de filtre. Les métadonnées ne sont pas indexées, mais peuvent être consultées dans le segment brut à l'aide de la console ou de l'API X-Ray. Toute personne à qui vous accordez un accès en lecture à X-Ray peut consulter ces données.

Lorsque vous avez un grand nombre de clients instrumentés dans votre code, un seul segment de demande peut contenir un grand nombre de sous-segments, un par appel effectué à l'aide d'un client instrumenté. Vous pouvez organiser et grouper les sous-segments en enveloppant les appels clients dans des [sous-segments personnalisés](#). Vous pouvez créer un sous-segment personnalisé pour une fonction complète ou une quelconque section du code, puis enregistrer les métadonnées et les annotations sur le sous-segment au lieu de tout écrire sur le segment parent.

Prérequis

Le SDK X-Ray pour Go nécessite Go 1.9 ou version ultérieure.

Le kit de développement logiciel dépend des bibliothèques suivantes lors de la compilation et de l'exécution :

- AWS SDK pour Go version 1.10.0 ou ultérieure

Ces dépendances sont déclarées dans le fichier README .md du kit de développement logiciel.

Documentation de référence

Une fois que vous avez téléchargé le kit SDK, générez et hébergez la documentation en local pour l'afficher dans un navigateur web.

Pour consulter la documentation de référence

1. Accédez au répertoire `$GOPATH/src/github.com/aws/aws-xray-sdk-go` (Linux ou Mac) ou au dossier `%GOPATH%\src\github.com\aws\aws-xray-sdk-go` (Windows)
2. Exécutez la commande `godoc`.

```
$ godoc -http=:6060
```

3. Ouvrez un navigateur à l'adresse `http://localhost:6060/pkg/github.com/aws/aws-xray-sdk-go/`.

Configuration du kit SDK X-Ray pour Go

Vous pouvez spécifier la configuration du SDK X-Ray pour Go par le biais de variables d'environnement, en Configure appelant avec Config un objet ou en utilisant des valeurs par défaut. Les variables d'environnement ont priorité sur les valeurs Config, qui prévalent elles-même sur n'importe quelle valeur par défaut.

Sections

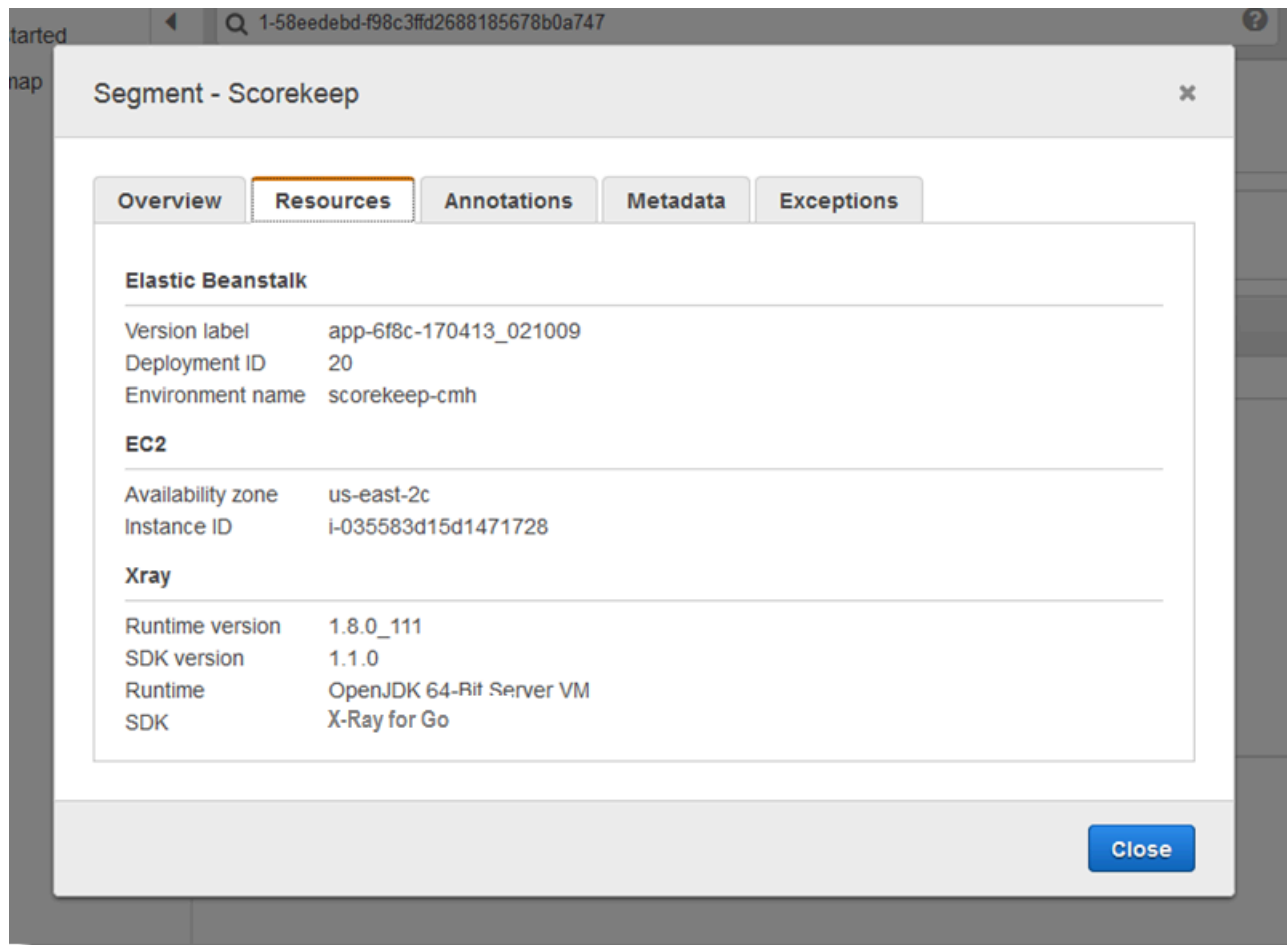
- [Plug-ins de service](#)
- [Règles d'échantillonnage](#)
- [Journalisation](#)
- [Variables d'environnement](#)
- [Utilisation de Configure](#)

Plug-ins de service

Permet plugins d'enregistrer des informations sur le service hébergeant votre application.

Plugins

- Amazon EC2 — EC2Plugin ajoute l'ID de l'instance, la zone de disponibilité et le groupe de CloudWatch journaux.
- Elastic ElasticBeanstalkPlugin Beanstalk : ajoute le nom de l'environnement, l'étiquette de version et l'ID de déploiement.
- Amazon ECS — ECSPlugin ajoute l'ID du conteneur.



Pour utiliser un module d'extension, importez l'un des modules suivants :

```
"github.com/aws/aws-xray-sdk-go/awsplugins/ec2"
"github.com/aws/aws-xray-sdk-go/awsplugins/ecs"
"github.com/aws/aws-xray-sdk-go/awsplugins/beanstalk"
```

Chaque plug-in a un appel de fonction `Init()` explicite qui charge le plug-in.

Exemple `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",
})
}
```

Le SDK utilise également les paramètres du plugin pour définir le `origin` champ du segment. Cela indique le type de AWS ressource qui exécute votre application. Lorsque vous utilisez plusieurs plugins, le SDK utilise l'ordre de résolution suivant pour déterminer l'origine : ElasticBeanstalk > EKS > ECS > EC2.

Règles d'échantillonnage

Le SDK utilise les règles d'échantillonnage que vous définissez dans la console X-Ray pour déterminer les demandes à enregistrer. La règle par défaut suit la première demande chaque seconde, et 5 % de toutes les demandes supplémentaires provenant de tous les services envoient des traces à X-Ray. [Créez des règles supplémentaires dans la console X-Ray](#) pour personnaliser la quantité de données enregistrées pour chacune de vos applications.

Le SDK applique les règles personnalisées dans l'ordre dans lequel elles sont définies. Si une demande correspond à plusieurs règles personnalisées, le SDK applique uniquement la première règle.

Note

Si le SDK ne parvient pas à accéder à X-Ray pour obtenir des règles d'échantillonnage, il revient à une règle locale par défaut concernant la première demande chaque seconde, et 5 % des demandes supplémentaires par hôte. Cela peut se produire si l'hôte n'est pas autorisé à appeler `sampling APIs` ou ne peut pas se connecter au daemon X-Ray, qui agit comme un proxy TCP pour les appels d'API effectués par le SDK.

Vous pouvez également configurer le SDK pour charger des règles d'échantillonnage à partir d'un document JSON. Le SDK peut utiliser les règles locales comme solution de rechange dans les cas où l'échantillonnage X-Ray n'est pas disponible, ou utiliser exclusivement les règles locales.

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
  }
}
```

Cet exemple définit une règle personnalisée et une règle par défaut. La règle personnalisée applique un taux d'échantillonnage de 5 % sans nombre minimum de demandes à suivre pour les chemins sous-jacents. `/api/move/` La règle par défaut suit la première demande chaque seconde et 10 % des demandes supplémentaires.

L'inconvénient de définir des règles localement est que la cible fixe est appliquée par chaque instance de l'enregistreur indépendamment, au lieu d'être gérée par le service X-Ray. Au fur et à mesure que vous déployez de nouveaux hôtes, le taux fixe est multiplié, ce qui complique le contrôle de la quantité de données enregistrées.

Activé AWS Lambda, vous ne pouvez pas modifier le taux d'échantillonnage. Si votre fonction est appelée par un service instrumenté, les appels ayant généré des demandes échantillonnées par ce service seront enregistrés par Lambda. Si le suivi actif est activé et qu'aucun en-tête de suivi n'est présent, Lambda prend la décision d'échantillonnage.

Pour fournir les règles de sauvegarde, pointez sur le fichier JSON d'échantillonnage local en utilisant `NewCentralizedStrategyWithFilePath`.

Example main.go — Règle d'échantillonnage local

```
s, _ := sampling.NewCentralizedStrategyWithFilePath("sampling.json") // path to local
sampling json
```

```
xray.Configure(xray.Config{SamplingStrategy: s})
```

Pour utiliser uniquement les règles locales, pointez sur le fichier JSON d'échantillonnage local en utilisant `NewLocalizedStrategyFromFilePath`.

Exemple `main.go` — Désactive l'échantillonnage

```
s, _ := sampling.NewLocalizedStrategyFromFilePath("sampling.json") // path to local
sampling json
xray.Configure(xray.Config{SamplingStrategy: s})
```

Journalisation

Note

Les champs `xray.Config{}` `LogLevel` et `LogFormat` sont obsolètes à partir de la version 1.0.0-rc.10.

X-Ray utilise l'interface suivante pour la journalisation. L'enregistreur d'événements par défaut écrit sur `stdout` pour `LogLevelInfo` et au-dessus.

```
type Logger interface {
    Log(level LogLevel, msg fmt.Stringer)
}

const (
    LogLevelDebug LogLevel = iota + 1
    LogLevelInfo
    LogLevelWarn
    LogLevelError
)
```

Exemple écrire à **`io.Writer`**

```
xray.SetLogger(xraylog.NewDefaultLogger(os.Stderr, xraylog.LogLevelError))
```

Variables d'environnement

Vous pouvez utiliser des variables d'environnement pour configurer le SDK X-Ray for Go. Le kit SDK prend en charge les variables suivantes.

- `AWS_XRAY_CONTEXT_MISSING`— Réglé sur `RUNTIME_ERROR` pour générer des exceptions lorsque votre code instrumenté tente d'enregistrer des données alors qu'aucun segment n'est ouvert.

Valeurs valides

- `RUNTIME_ERROR`— Lance une exception d'exécution.
- `LOG_ERROR`— Enregistrez une erreur et continuez (par défaut).
- `IGNORE_ERROR`— Ignorez l'erreur et continuez.

Des erreurs liées à des segments ou sous-segments manquants peuvent se produire lorsque vous essayez d'utiliser un client instrumenté dans un code de démarrage qui s'exécute lorsqu'aucune demande n'est ouverte, ou dans un code qui génère un nouveau thread.

- `AWS_XRAY_TRACING_NAME`— Définissez le nom de service que le SDK utilise pour les segments.
- `AWS_XRAY_DAEMON_ADDRESS`— Définissez l'hôte et le port de l'écouteur du daemon X-Ray. Par défaut, le SDK envoie les données de suivi à `127.0.0.1:2000`. Utilisez cette variable si vous avez configuré le démon pour qu'il [écoute sur un port différent](#) ou s'il s'exécute sur un autre hôte.
- `AWS_XRAY_CONTEXT_MISSING`— Définissez la valeur pour déterminer comment le SDK gère les erreurs de contexte manquantes. Des erreurs relatives aux segments ou aux sous-segments manquants peuvent se produire lorsque vous tentez d'utiliser un client instrumenté dans le code de démarrage quand aucune demande n'est ouverte, ou dans un code qui génère un nouveau thread.
 - `RUNTIME_ERROR`— Par défaut, le SDK est configuré pour lancer une exception d'exécution.
 - `LOG_ERROR`— Paramétré pour enregistrer une erreur et continuer.

Les variables d'environnement remplacent les valeurs équivalentes définies dans le code.

Utilisation de Configure

Vous pouvez également configurer le SDK X-Ray pour Go à l'aide `Configure` de cette méthode. `Configure` prend un argument, un `Config` objet, avec les champs facultatifs suivants.

DaemonAddr

Cette chaîne indique l'hôte et le port de l'écouteur du daemon X-Ray. Si elle n'est pas spécifiée, X-Ray utilise la valeur de la variable d'AWS_XRAY_DAEMON_ADDRESS environnement. Si cette valeur n'est pas définie, c'est « 127.0.0.1:2000 » qui est utilisé.

ServiceVersion

Cette chaîne définit la version du service. Si elle n'est pas spécifiée, X-Ray utilise la chaîne vide (« »).

SamplingStrategy

Cet objet `SamplingStrategy` définit les appels d'application qui sont suivis. Si ce n'est pas spécifié, X-Ray utilise un `LocalizedSamplingStrategy`, qui prend la stratégie telle que définie dans `xray/resources/DefaultSamplingRules.json`.

StreamingStrategy

Cet `StreamingStrategy` objet indique s'il faut diffuser un segment lorsqu'il `RequiresStreaming` renvoie `true`. Si ce n'est pas spécifié, X-Ray utilise un `DefaultStreamingStrategy` qui diffuse un segment échantillonné si le nombre de sous-segments est supérieur à 20.

ExceptionFormattingStrategy

Cet objet `ExceptionFormattingStrategy` définit la façon dont vous souhaitez gérer les diverses exceptions. Si ce n'est pas spécifié, X-Ray utilise un `DefaultExceptionFormattingStrategy` avec un type `XrayError` `oferror`, le message d'erreur et le stack trace.

Instrumentation des requêtes HTTP entrantes avec le SDK X-Ray for Go

Vous pouvez utiliser le SDK X-Ray pour suivre les requêtes HTTP entrantes traitées par votre application sur une EC2 instance Amazon EC2 ou Amazon ECS. AWS Elastic Beanstalk

Utilisez `xray.Handler` pour instrumenter les demandes HTTP entrantes. Le SDK X-Ray pour Go implémente l'interface de `http.Handler` bibliothèque Go standard dans `xray.Handler` la classe pour intercepter les requêtes Web. La classe `xray.Handler` englobe l'élément `http.Handler` fourni avec `xray.Capture` en utilisant le contexte de la demande, en analysant les en-têtes entrants et en ajoutant des en-têtes de réponse si nécessaire, puis elle définit les champs de suivi propres à HTTP.

Lorsque vous utilisez cette classe pour gérer les requêtes et réponses HTTP, le SDK X-Ray pour Go crée un segment pour chaque requête échantillonnée. Ce segment comprend la durée, la méthode et l'état de la demande HTTP. L'instrumentation supplémentaire crée des sous-segments sur ce segment.

Note

Pour les AWS Lambda fonctions, Lambda crée un segment pour chaque requête échantillonnée. Pour plus d'informations, consultez [AWS Lambda et AWS X-Ray](#).

L'exemple suivant intercepte les requêtes sur le port 8000 et renvoie « Hello ! » en guise de réponse. Il crée le segment myApp et instrumente des appels par le biais de n'importe quelle application.

Exemple 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)
}
```

Chaque segment porte un nom qui identifie votre application dans la carte des services. Le segment peut être nommé de manière statique ou vous pouvez configurer le SDK pour qu'il soit nommé dynamiquement en fonction de l'en-tête de l'hôte dans la demande entrante. La dénomination dynamique vous permet de regrouper les traces en fonction du nom de domaine indiqué dans la demande et d'appliquer un nom par défaut si le nom ne correspond pas au modèle attendu (par exemple, si l'en-tête de l'hôte est falsifié).

Demandes transmises

Si un équilibreur de charge ou un autre intermédiaire transmet une demande à votre application, X-Ray prend l'adresse IP du client depuis l'`X-Forwarded-For`-tête de la demande plutôt que depuis l'adresse IP source du paquet IP. L'adresse IP du client enregistrée pour une demande transférée peut être falsifiée, elle ne doit donc pas être fiable.

Lorsqu'une demande est transmise, le SDK définit un champ supplémentaire dans le segment pour l'indiquer. Si le segment contient le champ `x_forwarded_for` défini sur `true`, l'adresse IP du client a été extraite de l'`X-Forwarded-For`-tête de la requête HTTP.

Le gestionnaire crée un segment pour chaque demande entrante avec un bloc `http` contenant les informations suivantes :

- Méthode HTTP : GET, POST, PUT, DELETE, etc.
- Adresse du client : adresse IP du client qui a envoyé la demande.
- Code de réponse : code de réponse HTTP pour la demande terminée.
- Moment : heure de début (date de réception de la demande) et heure de fin (date d'envoi de la réponse).
- Agent utilisateur — Le formulaire `user-agent` de la demande.
- Longueur du contenu : `content-length` extrait de la réponse.

Configuration d'une stratégie d'attribution de noms de segment

AWS X-Ray utilise un nom de service pour identifier votre application et la distinguer des autres applications, bases de données, ressources externes APIs et AWS ressources utilisées par votre application. Lorsque le SDK X-Ray génère des segments pour les demandes entrantes, il enregistre le nom du service de votre application dans le [champ du nom](#) du segment.

Le SDK X-Ray peut nommer les segments d'après le nom d'hôte dans l'en-tête de la requête HTTP. Cependant, cet en-tête peut être falsifié, ce qui peut entraîner la création de nœuds inattendus dans votre carte de service. Pour éviter que le SDK ne nomme les segments de manière incorrecte en raison de demandes contenant des en-têtes d'hôte falsifiés, vous devez spécifier un nom par défaut pour les demandes entrantes.

Si votre application traite des demandes pour plusieurs domaines, vous pouvez configurer le SDK pour qu'il utilise une stratégie de dénomination dynamique afin de refléter cela dans les noms des segments. Une stratégie de dénomination dynamique permet au SDK d'utiliser le nom d'hôte pour les demandes qui correspondent à un modèle attendu et d'appliquer le nom par défaut aux demandes qui ne le sont pas.

Par exemple, il se peut qu'une seule application envoie des demandes à trois sous-domaines : `www.example.com`, `api.example.com`, et `static.example.com`. Vous pouvez utiliser une stratégie de dénomination dynamique avec le modèle `*.example.com` pour identifier les segments

de chaque sous-domaine avec un nom différent, ce qui permet d'obtenir trois nœuds de service sur la carte des services. Si votre application reçoit des demandes dont le nom d'hôte ne correspond pas au modèle, vous verrez un quatrième nœud sur la carte des services avec un nom de remplacement que vous spécifiez.

Pour utiliser le même nom pour tous les segments de la demande, spécifiez le nom de votre application lorsque vous créez le gestionnaire, comme indiqué dans les sections précédentes.

Note

Vous pouvez remplacer le nom de service par défaut que vous définissez avec la `AWS_XRAY_TRACING_NAME` variable d'environnement [???](#).

Une stratégie d'attribution de noms dynamique définit un modèle auquel doivent correspondre les noms d'hôte et un nom par défaut à utiliser si le nom d'hôte de la demande HTTP ne correspond pas au modèle. Pour nommer des segments de façon dynamique, utilisez `NewDynamicSegmentNamer` pour configurer le nom par défaut et le modèle correspondant.

Exemple `main.go`

Si le nom d'hôte dans la demande correspond au modèle `*.example.com`, utilisez le nom d'hôte. Dans le cas contraire, utilisez `MyApp`.

```
func main() {
    http.Handle("/", xray.Handler(xray.NewDynamicSegmentNamer("MyApp", "*.example.com"),
    http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
        w.Write([]byte("Hello!"))
    })))

    http.ListenAndServe(":8000", nil)
}
```

Suivi des appels du AWS SDK avec le SDK X-Ray pour Go

Lorsque votre application effectue des appels Services AWS pour stocker des données, écrire dans une file d'attente ou envoyer des notifications, le X-Ray SDK for Go suit les appels en aval [dans](#) des sous-segments. Les ressources tracées Services AWS et auxquelles vous accédez au sein de ces services (par exemple, un compartiment Amazon S3 ou une file d'attente Amazon SQS) apparaissent sous forme de nœuds en aval sur la carte de suivi de la console X-Ray.

Pour suivre les clients SDK AWS, enveloppez l'objet client avec l'appel `xray.AWS()` comme indiqué dans l'exemple suivant.

Exemple main.go

```
var dynamo *dynamodb.DynamoDB
func main() {
    dynamo = dynamodb.New(session.Must(session.NewSession()))
    xray.AWS(dynamo.Client)
}
```

Ensuite, lorsque vous utilisez le client SDK AWS, utilisez la version `withContext` de la méthode d'appel et transmettez-lui l'élément `context` à partir de l'objet `http.Request` transmis au [gestionnaire](#).

Exemple main.go — appel au SDK AWS

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

Pour tous les services, vous pouvez voir le nom de l'API appelée dans la console X-Ray. Pour un sous-ensemble de services, le SDK X-Ray ajoute des informations au segment afin d'améliorer la granularité de la carte des services.

Par exemple, lorsque vous passez un appel avec un client DynamoDB instrumenté, le SDK ajoute le nom de la table au segment pour les appels qui ciblent une table. Dans la console, chaque table apparaît sous la forme d'un nœud distinct dans la carte des services, avec un nœud DynamoDB générique pour les appels qui ne ciblent pas une table.

Exemple Sous-segment pour un appel à DynamoDB pour enregistrer un élément

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

Lorsque vous accédez aux ressources nommées, les appels vers les services suivants créent des nœuds supplémentaires dans la cartographie des services. Les appels qui ne ciblent pas de ressources spécifiques créent un nœud générique pour le service.

- Amazon DynamoDB — Nom de la table
- Amazon Simple Storage Service : nom du compartiment et de la clé
- Amazon Simple Queue Service — Nom de la file d'attente

Suivi des appels vers les services Web HTTP en aval avec le SDK X-Ray pour Go

Lorsque votre application effectue des appels vers des microservices ou du protocole HTTP public APIs, vous pouvez utiliser le `xray.Client` pour instrumenter ces appels en tant que sous-segments de votre application Go, comme indiqué dans l'exemple suivant, où `http-client` est un client HTTP.

Le client crée une copie superficielle du client HTTP fourni, par défaut `http.DefaultClient`, avec `roundtripper` encapsulé avec `xray.RoundTripper`

Exemple

<caption>main.go — Client HTTP</caption>

```
myClient := xray.Client(http-client)
```

<caption>main.go — Trace un appel HTTP en aval avec la bibliothèque `ctxhttp`</caption>

L'exemple suivant instrumente l'appel HTTP sortant avec la bibliothèque `ctxhttp` en utilisant `xray.Client`. `ctxhttp` peut être transmis depuis l'appel en amont. Cela garantit que le contexte de

segment existant est utilisé. Par exemple, X-Ray n'autorise pas la création d'un nouveau segment dans une fonction Lambda. Le contexte du segment Lambda existant doit donc être utilisé.

```
resp, err := ctxhttp.Get(ctx, xray.Client(nil), url)
```

Suivi des requêtes SQL avec le SDK X-Ray pour Go

Pour suivre les appels SQL vers MySQL ou PostgreSQL, remplacez les appels `sql.Open` par `xray.SQLContext`, comme illustré dans l'exemple suivant. À utiliser à la place des chaînes de configuration si possible.

Exemple `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  
}
```

Génération de sous-segments personnalisés avec le SDK X-Ray pour Go

Les sous-segments étendent le [segment](#) d'une trace avec des détails sur le travail effectué afin de répondre à une demande. Chaque fois que vous passez un appel avec un client instrumenté, le SDK X-Ray enregistre les informations générées dans un sous-segment. Vous pouvez créer des sous-segments supplémentaires pour regrouper d'autres sous-segments, pour mesurer les performances d'une section de code ou pour enregistrer des annotations et des métadonnées.

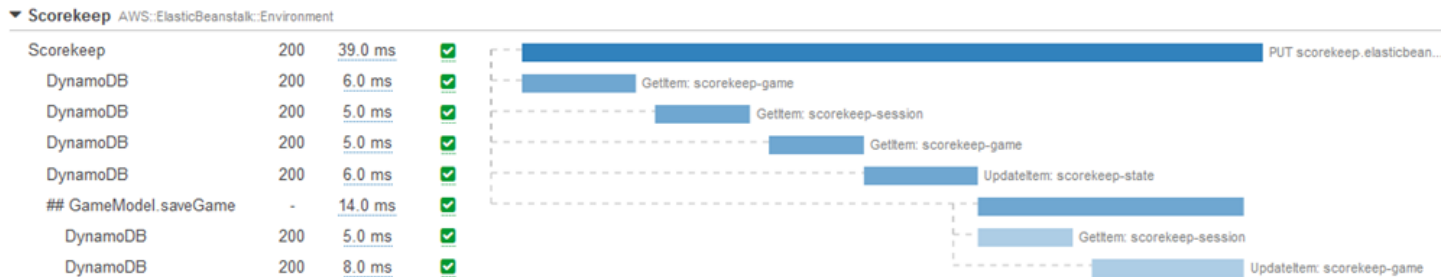
Pour créer un sous-segment pour une fonction, utilisez la méthode `Capture`.

Exemple `main.go` — Sous-segment personnalisé

```
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)
})
```

La capture d'écran suivante illustre un exemple de la façon dont le sous-segment `saveGame` peut apparaître dans les suivis de l'application `Scorekeep`.



Ajoutez des annotations et des métadonnées aux segments avec le SDK X-Ray pour Go

Vous pouvez enregistrer des informations supplémentaires sur les demandes, l'environnement ou votre application à l'aide d'annotations et de métadonnées. Vous pouvez ajouter des annotations et des métadonnées aux segments créés par le SDK X-Ray ou aux sous-segments personnalisés que vous créez.

Les annotations sont des paires clé-valeur avec des chaînes, des nombres ou des valeurs booléennes. Les annotations sont indexées pour être utilisées avec les expressions de [filtre](#). Utilisez les annotations pour enregistrer les données que vous souhaitez utiliser pour regrouper les suivis dans la console ou lors de l'appel de l'API [GetTraceSummaries](#).

Les métadonnées sont des paires clé-valeur qui peuvent contenir des valeurs de n'importe quel type, y compris des objets et des listes, mais qui ne sont pas indexées pour être utilisées avec des expressions de filtre. Utilisez les métadonnées pour enregistrer des données supplémentaires que vous souhaitez stocker dans le traçage, mais dont vous n'avez pas besoin pour effectuer une recherche.

En plus des annotations et des métadonnées, vous pouvez également [enregistrer les chaînes d'ID utilisateur](#) sur des segments. Les IDs Les utilisateurs sont enregistrés dans un champ distinct sur les segments et sont indexés pour être utilisés dans le cadre de la recherche.

Sections

- [Enregistrer des annotations avec le SDK X-Ray pour Go](#)

- [Enregistrement de métadonnées avec le SDK X-Ray pour Go](#)
- [Utilisateur enregistrant IDs avec le SDK X-Ray for Go](#)

Enregistrer des annotations avec le SDK X-Ray pour Go

Utilisez les annotations pour enregistrer les informations sur les segments qui doivent être indexés pour la recherche.

Exigences liées aux annotations

- Clés — La clé d'une annotation X-Ray peut comporter jusqu'à 500 caractères alphanumériques. Vous ne pouvez pas utiliser d'espaces ou de symboles autres qu'un point ou un point (.)
- Valeurs — La valeur d'une annotation X-Ray peut comporter jusqu'à 1 000 caractères Unicode.
- Nombre d'annotations : vous pouvez utiliser jusqu'à 50 annotations par trace.

Pour enregistrer des annotations, appelez `AddAnnotation` avec une chaîne contenant les métadonnées à associer au segment.

```
xray.AddAnnotation(key string, value interface{})
```

Le kit de développement logiciel enregistre les annotations sous forme de paires clé-valeur dans un objet `annotations` du document de segment. Si vous appelez deux fois `AddAnnotation` avec la même clé, les valeurs précédemment enregistrées sur le même segment sont remplacées.

Pour rechercher les suivis ayant des annotations avec des valeurs spécifiques, utilisez le mot clé `annotation[key]` dans une [expression de filtre](#).

Enregistrement de métadonnées avec le SDK X-Ray pour Go

Utilisez les métadonnées pour enregistrer des informations sur les segments qui n'ont pas besoin d'être indexés pour la recherche.

Pour enregistrer des métadonnées, appelez `AddMetadata` avec une chaîne contenant les métadonnées à associer au segment.

```
xray.AddMetadata(key string, value interface{})
```

Utilisateur enregistrant IDs avec le SDK X-Ray for Go

Enregistrez les segments utilisateur IDs sur demande pour identifier l'utilisateur qui a envoyé la demande.

Pour enregistrer un utilisateur IDs

1. Obtenez une référence au segment en cours à partir d'AWSXRay.

```
import (  
    "context"  
    "github.com/aws/aws-xray-sdk-go/xray"  
)  
  
mySegment := xray.GetSegment(context)
```

2. Appelez setUser avec l'ID de type chaîne de l'utilisateur ayant envoyé la demande.

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

Pour rechercher les suivis d'un ID utilisateur, utilisez le mot clé user dans une [expression de filtre](#).

Utilisation de Java

Il existe deux manières d'instrumenter votre application Java pour envoyer des traces à X-Ray :

- [AWS Distro pour OpenTelemetry Java](#) : AWS distribution qui fournit un ensemble de bibliothèques open source permettant d'envoyer des métriques et des traces corrélées à plusieurs solutions de AWS surveillance, notamment Amazon et Amazon OpenSearch Service CloudWatch AWS X-Ray, via [AWS Distro](#) for Collector. OpenTelemetry
- [AWS X-Ray SDK pour](#) Java : ensemble de bibliothèques permettant de générer et d'envoyer des traces à X-Ray via le daemon [X-Ray](#).

Pour de plus amples informations, veuillez consulter [Choisir entre The AWS Distro for OpenTelemetry et X-Ray SDKs](#).

AWS Distro pour Java OpenTelemetry

Avec AWS Distro for OpenTelemetry (ADOT) Java, vous pouvez instrumenter vos applications une seule fois et envoyer des métriques et des traces corrélées à plusieurs solutions de AWS surveillance CloudWatch, notamment Amazon AWS X-Ray et Amazon Service. OpenSearch L'utilisation de X-Ray avec ADOT nécessite deux composants : un OpenTelemetry SDK activé pour une utilisation avec X-Ray, et le AWS Distro for OpenTelemetry Collector activé pour une utilisation avec X-Ray. ADOT Java inclut la prise en charge de l'instrumentation automatique, permettant à votre application d'envoyer des traces sans modifier le code.

Pour commencer, consultez la [documentation de AWS Distro pour OpenTelemetry Java](#).

Pour plus d'informations sur l'utilisation de la AWS distribution pour OpenTelemetry with AWS X-Ray et d'autres Services AWS, consultez la section [AWS Distro for OpenTelemetry](#) ou la distribution pour la [AWS documentation](#). OpenTelemetry

Pour plus d'informations sur la prise en charge et l'utilisation des langues, consultez [AWS Observability on GitHub](#).

AWS X-Ray SDK pour Java

Le SDK X-Ray pour Java est un ensemble de bibliothèques pour les applications Web Java qui fournissent des classes et des méthodes permettant de générer et d'envoyer des données de trace

au daemon X-Ray. Les données de suivi incluent des informations sur les requêtes HTTP entrantes traitées par l'application et les appels que l'application envoie aux services en aval à l'aide du AWS SDK, des clients HTTP ou d'un connecteur de base de données SQL. Vous pouvez également créer des segments manuellement et ajouter des informations de débogage dans les annotations et les métadonnées.

Le SDK X-Ray pour Java est un projet open source. Vous pouvez suivre le projet et soumettre des problèmes et des pull requests sur GitHub : github.com/aws/aws-xray-sdk-java

Commencez par [ajouter AWSXRayServletFilter en tant que filtre servlet](#) pour suivre les demandes entrantes. Un filtre de servlet crée un [segment](#). Lorsque le segment est ouvert, vous pouvez utiliser les méthodes du client du SDK pour ajouter des informations au segment et créer des sous-segments afin de suivre les appels en aval. Le kit de développement logiciel (SDK) enregistre aussi automatiquement les exceptions que votre application renvoie quand le segment est ouvert.

À partir de la version 1.3, vous pouvez instrumenter votre application à l'aide de la [programmation orientée aspect \(AOP\) dans Spring](#). Cela signifie que vous pouvez instrumenter votre application pendant qu'elle est en cours d'exécution AWS, sans ajouter de code au runtime de votre application.

Utilisez ensuite le SDK X-Ray pour Java pour instrumenter AWS SDK pour Java vos clients [en incluant le sous-module SDK Instrumentor](#) dans votre configuration de compilation. Chaque fois que vous appelez une ressource Service AWS ou une ressource en aval avec un client instrumenté, le SDK enregistre les informations relatives à l'appel dans un sous-segment. Services AWS et les ressources auxquelles vous accédez au sein des services apparaissent sous forme de nœuds en aval sur la carte de trace pour vous aider à identifier les erreurs et les problèmes de limitation sur les connexions individuelles.

Si vous ne souhaitez pas instrumenter tous les appels en aval Services AWS, vous pouvez omettre le sous-module Instrumentor et choisir les clients à instrumenter. Instrumentez des clients individuels [en ajoutant un TracingHandler](#) à un client de service AWS SDK.

D'autres sous-modules du SDK X-Ray pour Java fournissent une instrumentation pour les appels en aval aux bases de données APIs Web HTTP et SQL. Vous pouvez [utiliser le SDK X-Ray pour les versions Java HTTPClient de HTTPClientBuilder](#) et dans le sous-module HTTP Apache pour instrumenter les clients HTTP Apache. Pour instrumenter les requêtes SQL, [ajoutez l'intercepteur du kit de développement logiciel \(SDK\) à votre source de données](#).

Après avoir commencé à utiliser le SDK, personnalisez son comportement en [configurant l'enregistreur et le filtre de servlet](#). Vous pouvez ajouter des plug-ins pour enregistrer les

données sur les ressources de calcul exécutant votre application, personnaliser le comportement d'échantillonnage en définissant des règles d'échantillonnage et définir le niveau de journal afin d'afficher plus ou moins d'informations du kit de développement logiciel dans vos journaux d'application.

Enregistrez les informations supplémentaires sur les demandes et le travail que votre application effectue dans les [annotations et les métadonnées](#). Les annotations sont de simples paires clé-valeur, indexées en vue de leur utilisation avec les [expressions de filtre](#), de telle sorte que vous pouvez explorer les suivis qui contiennent des données spécifiques. Les entrées des métadonnées sont moins restrictives et peuvent enregistrer des objets et tableaux entiers (tout ce qui peut être sérialisé en JSON).

Annotations et métadonnées

Les annotations et les métadonnées sont du texte arbitraire que vous ajoutez aux segments avec le kit de développement X-Ray. Les annotations sont indexées pour être utilisées avec les expressions de filtre. Les métadonnées ne sont pas indexées, mais peuvent être consultées dans le segment brut à l'aide de la console ou de l'API X-Ray. Toute personne à qui vous accordez un accès en lecture à X-Ray peut consulter ces données.

Lorsque vous avez un grand nombre de clients instrumentés dans votre code, un seul segment de demande peut contenir un grand nombre de sous-segments, un par appel effectué à l'aide d'un client instrumenté. Vous pouvez organiser et grouper les sous-segments en enveloppant les appels clients dans des [sous-segments personnalisés](#). Vous pouvez créer un sous-segment personnalisé pour une fonction complète ou une quelconque section du code, puis enregistrer les métadonnées et les annotations sur le sous-segment au lieu de tout écrire sur le segment parent.

Sous-modules

Vous pouvez télécharger le SDK X-Ray pour Java depuis Maven. Le SDK X-Ray pour Java est divisé en sous-modules par cas d'utilisation, avec une nomenclature pour la gestion des versions :

- [aws-xray-recorder-sdk-core](#)(obligatoire) — Fonctionnalité de base pour créer des segments et transmettre des segments. Inclut `AWSXRayServletFilter` pour instrumenter des demandes entrantes.
- [aws-xray-recorder-sdk-aws-sdk](#)— Instrumente les appels Services AWS passés avec AWS SDK pour Java les clients en ajoutant un client de suivi en tant que gestionnaire de demandes.

- [aws-xray-recorder-sdk-aws-sdk-v2](#)— Instrumente les appels Services AWS effectués avec les clients AWS SDK pour Java 2.2 et versions ultérieures en ajoutant un client de suivi comme intercepteur de demandes.
- [aws-xray-recorder-sdk-aws-sdk-instrumentor](#)— Avec `aws-xray-recorder-sdk-aws-sdk`, instrumente automatiquement tous les AWS SDK pour Java clients.
- [aws-xray-recorder-sdk-aws-sdk-v2-instrumentor](#)— Avec `aws-xray-recorder-sdk-aws-sdk-v2`, instrumente automatiquement tous les clients AWS SDK pour Java 2.2 et versions ultérieures.
- [aws-xray-recorder-sdk-apache-http](#)— Instruments les appels HTTP sortants effectués avec les clients HTTP Apache.
- [aws-xray-recorder-sdk-spring](#)— Fournit des intercepteurs pour les applications Spring AOP Framework.
- [aws-xray-recorder-sdk-sql-postgres](#)— Instruments les appels sortants vers une base de données PostgreSQL effectués avec JDBC.
- [aws-xray-recorder-sdk-sql-mysql](#)— Instruments les appels sortants vers une base de données MySQL effectués avec JDBC.
- [aws-xray-recorder-sdk-bom](#)— Fournit une nomenclature que vous pouvez utiliser pour spécifier la version à utiliser pour tous les sous-modules.
- [aws-xray-recorder-sdk-metrics](#)— Publiez des CloudWatch statistiques Amazon non échantillonnées à partir des segments X-Ray que vous avez collectés.

Si vous utilisez Maven ou Gradle pour créer votre application, [ajoutez le SDK X-Ray pour Java](#) à votre configuration de compilation.

Pour la documentation de référence sur les classes et les méthodes du SDK, consultez le manuel de référence des API du [AWS X-Ray SDK for Java](#).

Prérequis

Le SDK X-Ray pour Java nécessite Java 8 ou version ultérieure, l'API Servlet 3, AWS le SDK et Jackson.

Le kit de développement logiciel dépend des bibliothèques suivantes lors de la compilation et de l'exécution :

- AWS SDK pour Java version 1.11.398 ou ultérieure

- Servlet API 3.1.0

Ces dépendances sont déclarées dans le fichier `pom.xml` du kit de développement logiciel et sont incluses automatiquement si vous créez à l'aide de Maven ou Gradle.

Si vous utilisez une bibliothèque incluse dans le SDK X-Ray pour Java, vous devez utiliser la version incluse. Par exemple, si vous dépendez déjà de Jackson lors de l'exécution et incluez des fichiers JAR dans votre déploiement pour cette dépendance, vous devez supprimer ces fichiers JAR, car le fichier JAR du kit SDK inclut ses propres versions des bibliothèques Jackson.

Gestion des dépendances

Le SDK X-Ray pour Java est disponible auprès de Maven :

- Groupe — `com.amazonaws`
- Artifacts — `aws-xray-recorder-sdk-bom`
- Version — `2.11.0`

Si vous utilisez Maven pour créer votre application, ajoutez le kit de développement logiciel en tant que dépendance dans votre fichier `pom.xml`.

Exemple `pom.xml` - dépendances

```
<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>

```

Pour Gradle, ajoutez le kit de développement logiciel en tant que dépendance de compilation dans votre fichier `build.gradle`.

Exemple build.gradle - dépendances

```

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')
    }
}

```

```
}
```

Si vous utilisez Elastic Beanstalk pour déployer votre application, vous pouvez utiliser Maven ou Gradle pour créer sur instance chaque fois que vous déployez, au lieu de créer et de télécharger une archive volumineuse contenant toutes vos dépendances. Consultez l' [exemple d'application](#) pour obtenir un exemple utilisant Gradle.

AWS X-Ray agent d'auto-instrumentation pour Java

L'agent AWS X-Ray d'auto-instrumentation pour Java est une solution de traçage qui permet d'instrumenter vos applications Web Java avec un minimum d'efforts de développement. L'agent permet le suivi des applications basées sur des servlets et de toutes les demandes en aval de l'agent effectuées avec des frameworks et des bibliothèques pris en charge. Cela inclut les requêtes HTTP Apache en aval, les requêtes AWS SDK et les requêtes SQL effectuées à l'aide d'un pilote JDBC. L'agent propage le contexte X-Ray, y compris tous les segments et sous-segments actifs, entre les threads. Toutes les configurations et la polyvalence du SDK X-Ray sont toujours disponibles avec l'agent Java. Des valeurs par défaut appropriées ont été choisies pour garantir que l'agent fonctionne avec un minimum d'effort.

La solution X-Ray Agent convient parfaitement aux serveurs d'applications Web Java basés sur des servlets et basés sur des requêtes/réponses. Si votre application utilise un framework asynchrone ou n'est pas correctement modélisée en tant que service de demande-réponse, vous pouvez plutôt envisager une instrumentation manuelle avec le SDK.

L'agent X-Ray est conçu à l'aide de la boîte à outils Distributed Systems Compréhension, ou DiSCo. Di SCo est un framework open source permettant de créer des agents Java utilisables dans des systèmes distribués. Bien qu'il ne soit pas nécessaire de comprendre Di SCo pour utiliser l'agent X-Ray, vous pouvez en savoir plus sur le projet en visitant sa [page d'accueil sur GitHub](#). L'agent X-Ray est également entièrement open source. Pour consulter le code source, apporter des contributions ou signaler des problèmes concernant l'agent, consultez son [référentiel sur GitHub](#).

Exemple d'application

L'application [eb-java-scorekeep](#) d'échantillonnage est adaptée pour être instrumentée avec l'agent X-Ray. Cette branche ne contient aucune configuration de filtre de servlet ou d'enregistreur, car ces fonctions sont effectuées par l'agent. Pour exécuter l'application localement ou à l'aide de AWS ressources, suivez les étapes décrites dans le fichier readme de l'exemple d'application. Les instructions d'utilisation de l'exemple d'application pour générer des traces X-Ray se trouvent dans le [didacticiel de l'exemple d'application](#).

Premiers pas

Pour commencer à utiliser l'agent Java d'auto-instrumentation X-Ray dans votre propre application, procédez comme suit.

1. Exécutez le daemon X-Ray dans votre environnement. Pour de plus amples informations, veuillez consulter [AWS X-Ray démon](#).
2. Téléchargez la [dernière distribution de l'agent](#). Décompressez l'archive et notez son emplacement dans votre système de fichiers. Son contenu doit ressembler à ce qui suit.

```
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. Modifiez les arguments JVM de votre application pour inclure les éléments suivants, qui activent l'agent. Assurez-vous que l'-javaagentargument est placé avant l'-jarargument, le cas échéant. Le processus de modification des arguments JVM varie en fonction des outils et des frameworks que vous utilisez pour lancer votre serveur Java. Consultez la documentation de votre infrastructure de serveur pour obtenir des conseils spécifiques.

```
-javaagent:/<path-to-disco>/disco-java-agent.jar=pluginPath=/<path-to-disco>/disco-plugins
```

4. Pour spécifier la manière dont le nom de votre application apparaît sur la console X-Ray, définissez la variable d'AWS_XRAY_TRACING_NAMEenvironnement ou la propriété `com.amazonaws.xray.strategy.tracingName` système. Si aucun nom n'est fourni, un nom par défaut est utilisé.
5. Redémarrez votre serveur ou votre conteneur. Les demandes entrantes et leurs appels en aval sont désormais tracés. Si vous ne voyez pas les résultats escomptés, consultez [the section called "Résolution des problèmes"](#).

Configuration

L'agent X-Ray est configuré par un fichier JSON externe fourni par l'utilisateur. Par défaut, ce fichier se trouve à la racine du chemin de classe de l'utilisateur (par exemple, dans son `resources`

répertoire) nommé `xray-agent.json`. Vous pouvez configurer un emplacement personnalisé pour le fichier de configuration en définissant la propriété `com.amazonaws.xray.configFile` système sur le chemin absolu du système de fichiers de votre fichier de configuration.

Un exemple de fichier de configuration est présenté ci-dessous.

```
{
  "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
}
```

Spécification de configuration

Le tableau suivant décrit les valeurs valides pour chaque propriété. Les noms de propriétés distinguent les majuscules des minuscules, mais pas leurs clés. Pour les propriétés qui peuvent être remplacées par des variables d'environnement et des propriétés système, l'ordre de priorité est toujours variable d'environnement, puis propriété système, puis fichier de configuration. Pour plus d'informations sur les propriétés que vous pouvez remplacer, consultez [Variables d'environnement](#). Tous les champs sont facultatifs.

Nom de la propriété	Type	Valeurs valides	Description	Variable d'environnement	Propriété du système	Par défaut
serviceName	Chaîne	Toute chaîne	Le nom de votre service instrumenté	AWS_XRAY_NOM_TRACE	com.amazonaws.xray.strategy	XRayInstrumentedService

Nom de la propriété	Type	Valeurs valides	Description	Variable d'environnement	Propriété du système	Par défaut
			té tel qu'il apparaîtra dans la console X-Ray.		.tracingName	
contextMissingStrategy	Chaîne	LOG_ERROR, IGNORE_ERROR	L'action entreprise par l'agent lorsqu'il tente d'utiliser le contexte du segment X-Ray mais qu'aucune n'est présente.	AWS_XRAY_CONTEXT_MISSING	com.amazonaws.xray.strategy.contextMissingStrategy	LOG_ERROR
Adresse du démon	Chaîne	Adresse IP et port formatés, ou liste d'adresses TCP et UDP	Adresse utilisée par l'agent pour communiquer avec le daemon X-Ray.	AWS_XRAY_DAEMON_ADDRESS	Adresse com.amazonaws.xray.emit.daemon	127,0.0.1:2000

Nom de la propriété	Type	Valeurs valides	Description	Variable d'environnement	Propriété du système	Par défaut
Activé pour le traçage	Booléen	Vrai, Faux	Permet l'instrumentation par l'agent X-Ray.	AWS_XRAY_ACTIVÉ POUR LE TRAÇAGE	com.amazonaws.xray.tracingEnabled	TRUE
Stratégie d'échantillonnage	Chaîne	CENTRAL, LOCAL, AUCUN, TOUS	Stratégie d'échantillonnage utilisée par l'agent. ALL capture toutes les demandes, NONE ne capture aucune demande. Voir les règles d'échantillonnage .	N/A	N/A	CENTRAL
traceIdInjectionPrefix	Chaîne	Toute chaîne	Inclut le préfixe fourni avant l'injection du trace IDs dans les logs.	N/A	N/A	Aucun (chaîne vide)

Nom de la propriété	Type	Valeurs valides	Description	Variable d'environnement	Propriété du système	Par défaut
samplingRulesManifest	Chaîne	Un chemin de fichier absolu	Chemin d'accès à un fichier de règles d'échantillonnage personnalisé à utiliser comme source des règles d'échantillonnage pour la stratégie d'échantillonnage locale ou des règles de secours pour la stratégie centrale.	N/A	N/A	DefaultSamplingRules.json

Nom de la propriété	Type	Valeurs valides	Description	Variable d'environnement	Propriété du système	Par défaut
awsServiceHandlerManifeste	Chaîne	Un chemin de fichier absolu	Le chemin d'accès à une liste d'autorisation de paramètres personnalisée, qui capture des informations supplémentaires provenant des clients du AWS SDK.	N/A	N/A	DefaultOperationParameterWhitelist.json
awsSdkVersion	Entier	1, 2	Version du AWS SDK pour Java que vous utilisez. Ignoré s'awsServiceHandlerManifeste n'est pas également défini.	N/A	N/A	2

Nom de la propriété	Type	Valeurs valides	Description	Variable d'environnement	Propriété du système	Par défaut
maxStackTraceLongueur	Entier	Entiers non négatifs	Nombre maximal de lignes d'une pile à enregistrer dans une trace.	N/A	N/A	50
Seuil de diffusion	Entier	Entiers non négatifs	Une fois qu'au moins autant de sous-segments sont fermés, ils sont transmis au daemon pour out-of-band éviter que les segments ne soient trop volumineux.	N/A	N/A	100

Nom de la propriété	Type	Valeurs valides	Description	Variable d'environnement	Propriété du système	Par défaut
tracedInjection	Booléen	Vrai, Faux	Permet l'injection d'un identifiant de trace X-Ray dans les journaux si les dépendances et la configuration décrites dans la configuration de journalisation sont également ajoutées. Sinon, ne fait rien.	N/A	N/A	TRUE

Nom de la propriété	Type	Valeurs valides	Description	Variable d'environnement	Propriété du système	Par défaut
Plugins activés	Booléen	Vrai, Faux	Active les plug-ins qui enregistrent les métadonnées relatives AWS aux environnements dans lesquels vous travaillez. Voir plugins .	N/A	N/A	TRUE
collectSqlQueries	Booléen	Vrai, Faux	Enregistre au mieux les chaînes de requête SQL dans des sous-segments SQL.	N/A	N/A	FALSE

Nom de la propriété	Type	Valeurs valides	Description	Variable d'environnement	Propriété du système	Par défaut
Propagation du contexte	Booléen	Vrai, Faux	Propage automatiquement le contexte X-Ray entre les threads si c'est vrai. Sinon, utilise Thread Local pour stocker le contexte et une propagation manuelle entre les threads est requise.	N/A	N/A	TRUE

Configuration de journalisation

Le niveau de journalisation de l'agent X-Ray peut être configuré de la même manière que le SDK X-Ray pour Java. Consultez [Journalisation](#) pour plus d'informations sur la configuration de la journalisation avec le SDK X-Ray pour Java.

Instrumentation manuelle

Si vous souhaitez effectuer une instrumentation manuelle en plus de l'instrumentation automatique de l'agent, ajoutez le SDK X-Ray en tant que dépendance à votre projet. Notez que les filtres de servlet personnalisés du SDK mentionnés dans [Tracing Incoming Requests](#) ne sont pas compatibles avec l'agent X-Ray.

 Note

Vous devez utiliser la dernière version du SDK X-Ray pour effectuer une instrumentation manuelle tout en utilisant l'agent.

Si vous travaillez dans un projet Maven, ajoutez les dépendances suivantes à votre `pom.xml` fichier.

```
<dependencies>
  <dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-xray-recorder-sdk-core</artifactId>
    <version>2.11.0</version>
  </dependency>
</dependencies>
```

Si vous travaillez dans un projet Gradle, ajoutez les dépendances suivantes à votre `build.gradle` fichier.

```
implementation 'com.amazonaws:aws-xray-recorder-sdk-core:2.11.0'
```

Vous pouvez ajouter [des sous-segments personnalisés](#) en plus [des annotations, des métadonnées et des utilisateurs IDs](#) lorsque vous utilisez l'agent, comme vous le feriez avec le SDK normal. L'agent propage automatiquement le contexte entre les threads. Aucune solution de contournement pour propager le contexte ne devrait donc être nécessaire lorsque vous travaillez avec des applications multithread.

Résolution des problèmes

Comme l'agent propose une instrumentation entièrement automatique, il peut être difficile d'identifier la cause première d'un problème lorsque vous rencontrez des problèmes. Si l'agent X-Ray ne fonctionne pas comme prévu, passez en revue les problèmes et solutions suivants. L'agent X-Ray et le SDK utilisent Jakarta Commons Logging (JCL). Pour voir le résultat de journalisation, assurez-vous qu'un pont reliant JCL à votre backend de journalisation se trouve sur le chemin de classe, comme dans l'exemple suivant : ou. `log4j-jcl` `jcl-over-slf4j`

Problème : j'ai activé l'agent Java sur mon application mais je ne vois rien sur la console X-Ray

Le daemon X-Ray s'exécute-t-il sur la même machine ?

Si ce n'est pas le cas, consultez la [documentation du daemon X-Ray](#) pour le configurer.

Dans les journaux de vos applications, voyez-vous un message tel que « Initializing the X-Ray Agent Recorder » ?

Si vous avez correctement ajouté l'agent à votre application, ce message est enregistré au niveau INFO au démarrage de votre application, avant qu'elle ne commence à recevoir des demandes. Si ce message n'est pas présent, cela signifie que l'agent Java n'est pas en cours d'exécution avec votre processus Java. Assurez-vous d'avoir suivi correctement toutes les étapes de configuration, sans fautes de frappe.

Dans les journaux de votre application, voyez-vous plusieurs messages d'erreur tels que « Suppression d'une exception manquante AWS X-Ray dans le contexte » ?

Ces erreurs se produisent parce que l'agent essaie d'instrumenter les requêtes en aval, telles que les requêtes du AWS SDK ou les requêtes SQL, mais il n'a pas réussi à créer automatiquement un segment. Si vous constatez un grand nombre de ces erreurs, l'agent n'est peut-être pas le meilleur outil pour votre cas d'utilisation et vous devriez plutôt envisager une instrumentation manuelle avec le kit de développement X-Ray. Vous pouvez également activer les [journaux de débogage](#) du SDK X-Ray pour voir la trace de l'endroit où se produisent les exceptions manquantes au contexte. Vous pouvez envelopper ces parties de votre code avec des segments personnalisés, ce qui devrait résoudre ces erreurs. Pour un exemple d'encapsulation de requêtes en aval avec des segments personnalisés, consultez l'exemple de code de la section [Instrumentation du code de démarrage](#).

Problème : certains des segments que je prévois n'apparaissent pas sur la console X-Ray

Votre application utilise-t-elle le multithreading ?

Si certains segments que vous pensez être créés n'apparaissent pas dans votre console, cela peut être dû aux fils d'arrière-plan de votre application. Si votre application exécute des tâches à l'aide de threads d'arrière-plan qui sont « déclenchés et oubliés », par exemple en effectuant un appel ponctuel à une fonction Lambda avec AWS le SDK ou en interrogeant régulièrement un point de terminaison HTTP, cela peut semer la confusion chez l'agent lorsqu'il propage le contexte entre les threads. Pour vérifier que c'est bien votre problème, activez les journaux de débogage du SDK X-Ray et vérifiez la présence de messages tels que : Pas d'émission de segment nommé car il contient des sous-segments en <NAME >cours. Pour contourner ce problème, vous pouvez essayer de joindre les fils d'arrière-plan avant le retour de votre serveur afin de vous assurer que tout le travail effectué dans ceux-ci est enregistré. Vous pouvez également définir la `contextPropagation` configuration de l'agent `false` pour désactiver la propagation du contexte dans les fils d'arrière-plan. Dans ce cas,

vous devrez instrumenter manuellement ces fils avec des segments personnalisés ou ignorer les exceptions manquantes liées au contexte qu'ils produisent.

Avez-vous défini des règles d'échantillonnage ?

Si des segments apparemment aléatoires ou inattendus apparaissent sur la console X-Ray, ou si les segments que vous pensez ne pas apparaître sur la console ne le sont pas, il se peut que vous rencontriez un problème d'échantillonnage. L'agent X-Ray applique un échantillonnage centralisé à tous les segments qu'il crée, en utilisant les règles de la console X-Ray. La règle par défaut est qu'un segment par seconde, plus 5 % des segments suivants sont échantillonnés. Cela signifie que les segments créés rapidement avec l'agent risquent de ne pas être échantillonnés. Pour résoudre ce problème, vous devez créer des règles d'échantillonnage personnalisées sur la console X-Ray afin d'échantillonner de manière appropriée les segments souhaités. Pour plus d'informations, voir [échantillonnage](#).

Configuration du SDK X-Ray pour Java

Le SDK X-Ray pour Java inclut une classe `AWSXRay` nommée qui fournit l'enregistreur global. Ceci est un `TracingHandler` que vous pouvez utiliser pour outiller votre code. Vous pouvez configurer l'enregistreur mondial pour personnaliser le `AWSXRayServletFilter` qui crée des segments pour les appels HTTP entrants.

Sections

- [Plug-ins de service](#)
- [Règles d'échantillonnage](#)
- [Journalisation](#)
- [Écouteurs de segment](#)
- [Variables d'environnement](#)
- [Propriétés système](#)

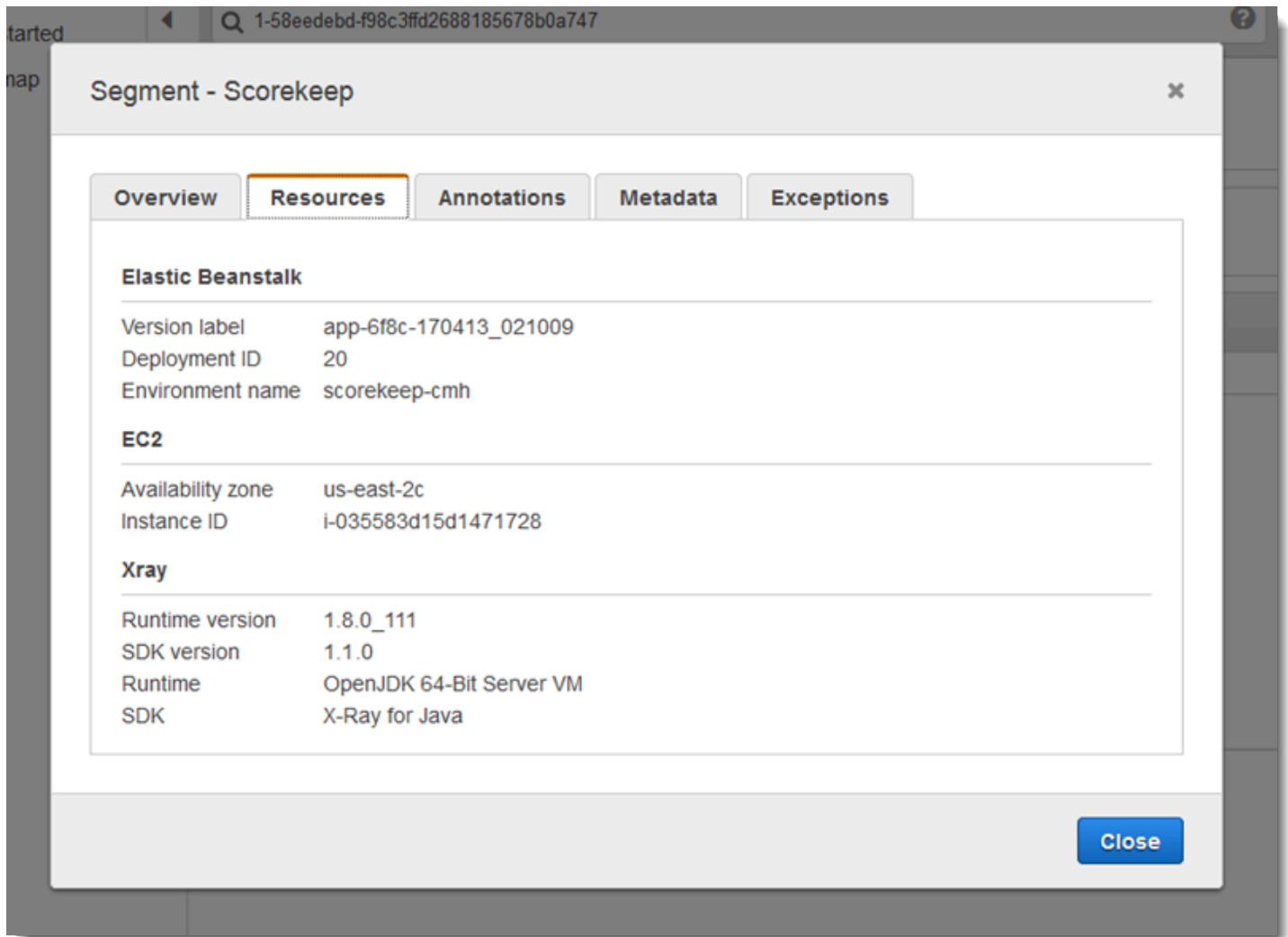
Plug-ins de service

Permet plugins d'enregistrer des informations sur le service hébergeant votre application.

Plugins

- Amazon EC2 — `EC2Plugin` ajoute l'ID de l'instance, la zone de disponibilité et le groupe de CloudWatch journaux.

- Elastic BeanstalkPlugin Beanstalk : ajoute le nom de l'environnement, l'étiquette de version et l'ID de déploiement.
- Amazon ECS — ECSPlugin ajoute l'ID du conteneur.
- Amazon EKS : EKSPugin ajoute l'ID du conteneur, le nom du cluster, l'ID du pod et le groupe de CloudWatch journaux.



Pour utiliser un plug-in, appelez `withPlugin` sur votre `AWSXRayRecorderBuilder`.

Exemple `src/main/java/scorekeep/WebConfig.java` - enregistreur

```
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());
    }
}
```

Le SDK utilise également les paramètres du plugin pour définir le `origin` champ du segment. Cela indique le type de AWS ressource qui exécute votre application. Lorsque vous utilisez plusieurs plugins, le SDK utilise l'ordre de résolution suivant pour déterminer l'origine : ElasticBeanstalk > EKS > ECS > EC2.

Règles d'échantillonnage

Le SDK utilise les règles d'échantillonnage que vous définissez dans la console X-Ray pour déterminer les demandes à enregistrer. La règle par défaut suit la première demande chaque seconde, et 5 % de toutes les demandes supplémentaires provenant de tous les services envoient des traces à X-Ray. [Créez des règles supplémentaires dans la console X-Ray](#) pour personnaliser la quantité de données enregistrées pour chacune de vos applications.

Le SDK applique les règles personnalisées dans l'ordre dans lequel elles sont définies. Si une demande correspond à plusieurs règles personnalisées, le SDK applique uniquement la première règle.

Note

Si le SDK ne parvient pas à accéder à X-Ray pour obtenir des règles d'échantillonnage, il revient à une règle locale par défaut concernant la première demande chaque seconde, et 5 % des demandes supplémentaires par hôte. Cela peut se produire si l'hôte n'est pas autorisé à appeler `sampling APIs` ou ne peut pas se connecter au daemon X-Ray, qui agit comme un proxy TCP pour les appels d'API effectués par le SDK.

Vous pouvez également configurer le SDK pour charger des règles d'échantillonnage à partir d'un document JSON. Le SDK peut utiliser les règles locales comme solution de rechange dans les cas où l'échantillonnage X-Ray n'est pas disponible, ou utiliser exclusivement les règles locales.

Exemple `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
  }
}
```

Cet exemple définit une règle personnalisée et une règle par défaut. La règle personnalisée applique un taux d'échantillonnage de 5 % sans nombre minimum de demandes à suivre pour les chemins sous-jacents. `/api/move/` La règle par défaut suit la première demande chaque seconde et 10 % des demandes supplémentaires.

L'inconvénient de définir des règles localement est que la cible fixe est appliquée par chaque instance de l'enregistreur indépendamment, au lieu d'être gérée par le service X-Ray. Au fur et à mesure que vous déployez de nouveaux hôtes, le taux fixe est multiplié, ce qui complique le contrôle de la quantité de données enregistrées.

Activé AWS Lambda, vous ne pouvez pas modifier le taux d'échantillonnage. Si votre fonction est appelée par un service instrumenté, les appels ayant généré des demandes échantillonnées par ce service seront enregistrés par Lambda. Si le suivi actif est activé et qu'aucun en-tête de suivi n'est présent, Lambda prend la décision d'échantillonnage.

Pour fournir des règles de sauvegarde dans Spring, configurez l'enregistreur mondial avec une stratégie `CentralizedSamplingStrategy` dans une classe de configuration.

Exemple src/main/java/myapp/WebConfig.java - configuration de l'enregistreur

```
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());
    }
}
```

Pour Tomcat, ajoutez un écouteur qui étend `ServletContextListener` et enregistrez l'écouteur dans le descripteur de déploiement.

Exemple 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) { }
}
```

Exemple WEB-INF/web.xml

```
...
<listener>
  <listener-class>com.myapp.web.Startup</listener-class>
</listener>
```

Pour utiliser les règles local uniquement, remplacez `CentralizedSamplingStrategy` par `LocalizedSamplingStrategy`.

```
builder.withSamplingStrategy(new LocalizedSamplingStrategy(ruleFile));
```

Journalisation

Par défaut, le SDK envoie des messages de ERROR niveau B dans les journaux de vos applications. Vous pouvez activer la journalisation au niveau du débogage sur le SDK pour générer des journaux plus détaillés dans le fichier journal de votre application. Les niveaux de journalisation valides sont DEBUG INFOWARN,,ERROR, etFATAL. FATALlog level désactive tous les messages de journal car le SDK ne se journalise pas à un niveau fatal.

Exemple application.properties

Définissez le niveau de journalisation avec la propriété `logging.level.com.amazonaws.xray`.

```
logging.level.com.amazonaws.xray = DEBUG
```

Utilisez les journaux de débogage pour identifier les problèmes, tels que des sous-segments ouverts, lorsque vous [générez manuellement des sous-segments](#).

Injection d'ID de suivi dans les journaux

Pour exposer l'ID de trace complet actuel à vos instructions de journal, vous pouvez l'injecter dans le contexte de diagnostic mappé (MDC). À l'aide de l'interface `SegmentListener`, les méthodes sont appelées à partir de l'enregistreur X-Ray lors des événements du cycle de vie des segments. Lorsqu'un segment ou un sous-segment commence, l'ID de trace qualifié est injecté dans le MDC avec la clé `AWS-XRAY-TRACE-ID`. Lorsque ce segment se termine, la clé est supprimée du MDC. Ceci expose l'ID de trace à la bibliothèque de journalisation en cours d'utilisation. Lorsqu'un sous-segment se termine, son ID parent est injecté dans le MDC.

Exemple ID de trace complet

L'ID complet est affiché au format `TraceID@EntityID`

```
1-5df42873-011e96598b447dfca814c156@541b3365be3dafc3
```

Cette fonctionnalité fonctionne avec les applications Java instrumentées avec le SDK AWS X-Ray pour Java et prend en charge les configurations de journalisation suivantes :

- SLF4API frontale J avec backend Logback
- SLF4API frontale J avec backend Log4J2
- API frontale Log4J2 avec backend Log4J2

Consultez les onglets suivants pour connaître les besoins de chaque frontal et de chaque backend.

SLF4J Frontend

1. Ajoutez la dépendance Maven suivante à votre projet.

```
<dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>aws-xray-recorder-sdk-slf4j</artifactId>
  <version>2.11.0</version>
</dependency>
```

2. Incluez la méthode `withSegmentListener` lors de la construction du `AWSXRayRecorder`. Cela ajoute une `SegmentListener` classe qui injecte automatiquement une nouvelle trace IDs dans le SLF4 J MDC.

SegmentListener utilise une chaîne facultative comme paramètre pour configurer le préfixe de l'instruction de journalisation. Le préfixe peut être configuré comme suit :

- Aucun — Utilise le AWS-XRAY-TRACE-ID préfixe par défaut.
- Vide — Utilise une chaîne vide (par exemple "").
- Personnalisé — Utilise un préfixe personnalisé tel que défini dans la chaîne.

Exemple **AWSXRayRecorderBuilder** déclaration

```
AWSXRayRecorderBuilder builder = AWSXRayRecorderBuilder
    .standard().withSegmentListener(new SLF4JSegmentListener("CUSTOM-
    PREFIX"));
```

Log4J2 front end

1. Ajoutez la dépendance Maven suivante à votre projet.

```
<dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>aws-xray-recorder-sdk-log4j</artifactId>
  <version>2.11.0</version>
</dependency>
```

2. Incluez la méthode withSegmentListener lors de la construction du AWSXRayRecorder. Cela ajoutera une SegmentListener classe qui injecte automatiquement une nouvelle trace entièrement qualifiée IDs dans le SLF4 J MDC.

SegmentListener utilise une chaîne facultative comme paramètre pour configurer le préfixe de l'instruction de journalisation. Le préfixe peut être configuré comme suit :

- Aucun — Utilise le AWS-XRAY-TRACE-ID préfixe par défaut.
- Vide — Utilise une chaîne vide (par exemple "") et supprime le préfixe.
- Personnalisé — Utilise le préfixe personnalisé défini dans la chaîne.

Exemple **AWSXRayRecorderBuilder** déclaration

```
AWSXRayRecorderBuilder builder = AWSXRayRecorderBuilder
```

```
.standard().withSegmentListener(new Log4JSegmentListener("CUSTOM-PREFIX"));
```

Logback backend

Pour insérer l'ID de trace dans vos événements de journal, vous devez modifier le `PatternLayout` de l'enregistreur, qui formate chaque instruction de journalisation.

1. Recherchez l'emplacement où `patternLayout` est configuré. Vous pouvez le faire par programme ou via un fichier de configuration XML. Pour en savoir plus, consultez [Configuration Logback](#).
2. Insérez `%X{AWS-XRAY-TRACE-ID}` n'importe où dans le `patternLayout` pour insérer l'ID de trace dans les instructions de journalisation futures. `%X{}` indique que vous récupérez une valeur avec la clé fournie à partir du MDC. Pour en savoir plus sur `PatternLayouts` Logback, voir [PatternLayout](#).

Log4J2 backend

1. Recherchez l'emplacement où `patternLayout` est configuré. Vous pouvez le faire par programme ou via un fichier de configuration écrit au format XML, JSON, YAML ou propriétés.

Pour en savoir plus sur la configuration de Log4J2 via un fichier de configuration, consultez [Configuration](#).

Pour en savoir plus sur la configuration de Log4J2 par programmation, consultez [Configuration programmatique](#).

2. Insérez `%X{AWS-XRAY-TRACE-ID}` n'importe où dans le `PatternLayout` pour insérer l'ID de trace dans les instructions de journalisation futures. `%X{}` indique que vous récupérez une valeur avec la clé fournie à partir du MDC. [Pour en savoir plus sur PatternLayouts Log4J2, consultez Pattern Layout](#).

Exemple d'injection d'ID de trace

Ce qui suit montre une chaîne `PatternLayout` modifiée pour inclure l'ID de trace. L'ID de trace est imprimé après le nom du thread (`%t`) et avant le niveau du journal (`%-5p`).

Exemple **PatternLayout** avec injection d'ID

```
%d{HH:mm:ss.SSS} [%t] %X{AWS-XRAY-TRACE-ID} %-5p %m%n
```

AWS X-Ray imprime automatiquement la clé et l'identifiant de trace dans l'instruction du journal pour faciliter l'analyse. Ce qui suit montre une instruction de journal à l'aide de la modification de **PatternLayout**.

Exemple Instruction de journalisation avec injection d'ID

```
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
```

Le message de journalisation lui-même est hébergé dans le modèle `%m` et est défini lors de l'appel de l'enregistreur.

Écouteurs de segment

Les écouteurs de segments sont une interface permettant d'intercepter les événements du cycle de vie tels que le début et la fin des segments produits par le `AWSXRayRecorder`. L'implémentation d'une fonction d'événement d'écouteur de segment peut consister à ajouter la même annotation à tous les sous-segments lorsqu'ils sont créés avec [onBeginSubsegment](#), à journaliser un message une fois que chaque segment est envoyé au démon à l'aide de [afterEndSegment](#), ou à enregistrer les requêtes envoyées par les intercepteurs SQL à l'aide de [beforeEndSubsegment](#) pour vérifier si le sous-segment représente une requête SQL, en ajoutant des métadonnées supplémentaires dans ce cas.

Pour consulter la liste complète des `SegmentListener` fonctions, consultez la documentation de l'API [AWS X-Ray Recorder SDK for Java](#).

L'exemple suivant montre comment ajouter une annotation cohérente à tous les sous-segments lors de la création avec [onBeginSubsegment](#) et comment imprimer un message de journal à la fin de chaque segment avec [afterEndSegment](#).

Exemple `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());
    }
}
```

Cet écouteur de segment personnalisé est ensuite référencé lors de la génération de `AWSXRayRecorder`.

Exemple `AWSXRayRecorderBuilder` déclaration

```
AWSXRayRecorderBuilder builder = AWSXRayRecorderBuilder
    .standard().withSegmentListener(new MySegmentListener());
```

Variables d'environnement

Vous pouvez utiliser des variables d'environnement pour configurer le SDK X-Ray pour Java. Le kit SDK prend en charge les variables suivantes.

- `AWS_XRAY_CONTEXT_MISSING`— Réglé sur `RUNTIME_ERROR` pour générer des exceptions lorsque votre code instrumenté tente d'enregistrer des données alors qu'aucun segment n'est ouvert.

Valeurs valides

- `RUNTIME_ERROR`— Lance une exception d'exécution.
- `LOG_ERROR`— Enregistrez une erreur et continuez (par défaut).
- `IGNORE_ERROR`— Ignorez l'erreur et continuez.

Des erreurs liées à des segments ou sous-segments manquants peuvent se produire lorsque vous essayez d'utiliser un client instrumenté dans un code de démarrage qui s'exécute lorsqu'aucune demande n'est ouverte, ou dans un code qui génère un nouveau thread.

- `AWS_XRAY_DAEMON_ADDRESS`— Définissez l'hôte et le port de l'écouteur du daemon X-Ray. Par défaut, le SDK utilise à la fois `127.0.0.1:2000` les données de trace (UDP) et l'échantillonnage (TCP). Utilisez cette variable si vous avez configuré le démon pour qu'il [écoute sur un port différent](#) ou s'il s'exécute sur un autre hôte.

Format

- Même port — *address:port*
- Différents ports — `tcp:address:port` `udp:address:port`
- `AWS_LOG_GROUP`— Définissez le nom d'un groupe de journaux sur le groupe de journaux associé à votre application. Si votre groupe de journaux utilise le même AWS compte et la même région que votre application, X-Ray recherchera automatiquement les données de segment de votre application à l'aide de ce groupe de journaux spécifié. Pour plus d'informations sur les groupes de journaux, consultez la section [Utilisation des groupes de journaux et des flux](#).
- `AWS_XRAY_TRACING_NAME`— Définissez un nom de service que le SDK utilise pour les segments. Remplace le nom de service que vous définissez sur la [stratégie d'attribution de noms de segment](#) du filtre servlet.

Les variables d'environnement remplacent les [propriétés de système](#) et les valeurs équivalentes définies dans le code.

Propriétés système

Vous pouvez utiliser les propriétés de système en tant qu'alternative JVM aux [variables d'environnement](#). Le kit SDK prend en charge les propriétés suivantes :

- `com.amazonaws.xray.strategy.tracingName`— Équivalent à `AWS_XRAY_TRACING_NAME`.
- `com.amazonaws.xray.emitters.daemonAddress`— Équivalent à `AWS_XRAY_DAEMON_ADDRESS`.
- `com.amazonaws.xray.strategy.contextMissingStrategy`— Équivalent à `AWS_XRAY_CONTEXT_MISSING`.

Si une propriété système et la variable d'environnement équivalente sont toutes les deux définies, c'est la valeur de la variable d'environnement qui est utilisée. Les deux méthodes remplacent les valeurs définies dans le code.

Suivi des demandes entrantes avec le SDK X-Ray pour Java

Vous pouvez utiliser le SDK X-Ray pour suivre les requêtes HTTP entrantes traitées par votre application sur une EC2 instance Amazon EC2 ou Amazon ECS. AWS Elastic Beanstalk

Utilisez un `Filter` pour instrumenter les demandes HTTP entrantes. Lorsque vous ajoutez le filtre de servlet X-Ray à votre application, le SDK X-Ray pour Java crée un segment pour chaque requête échantillonnée. Ce segment comprend la durée, la méthode et l'état de la demande HTTP. L'instrumentation supplémentaire crée des sous-segments sur ce segment.

Note

Pour les AWS Lambda fonctions, Lambda crée un segment pour chaque requête échantillonnée. Pour plus d'informations, consultez [AWS Lambda et AWS X-Ray](#).

Chaque segment porte un nom qui identifie votre application dans la carte des services. Le segment peut être nommé de manière statique ou vous pouvez configurer le SDK pour qu'il soit nommé dynamiquement en fonction de l'en-tête de l'hôte dans la demande entrante. La dénomination dynamique vous permet de regrouper les traces en fonction du nom de domaine indiqué dans la demande et d'appliquer un nom par défaut si le nom ne correspond pas au modèle attendu (par exemple, si l'en-tête de l'hôte est falsifié).

Demandes transmises

Si un équilibreur de charge ou un autre intermédiaire transmet une demande à votre application, X-Ray prend l'adresse IP du client depuis l'`X-Forwarded-For`-tête de la demande plutôt que depuis l'adresse IP source du paquet IP. L'adresse IP du client enregistrée pour une demande transférée peut être falsifiée, elle ne doit donc pas être fiable.

Lorsqu'une demande est transmise, le SDK définit un champ supplémentaire dans le segment pour l'indiquer. Si le segment contient le champ `x_forwarded_for` défini sur `true`, l'adresse IP du client a été extraite de l'`X-Forwarded-For`-tête de la requête HTTP.

Le gestionnaire de messages crée un segment pour chaque demande entrante avec un bloc `http` contenant les informations suivantes :

- Méthode HTTP : GET, POST, PUT, DELETE, etc.

- Adresse du client : adresse IP du client qui a envoyé la demande.
- Code de réponse : code de réponse HTTP pour la demande terminée.
- Moment : heure de début (date de réception de la demande) et heure de fin (date d'envoi de la réponse).
- Agent utilisateur — Le formulaire user-agent de la demande.
- Longueur du contenu : content-length extrait de la réponse.

Sections

- [Ajout d'un filtre de suivi à votre application \(Tomcat\)](#)
- [Ajout d'un filtre de suivi à votre application \(Spring\)](#)
- [Configuration d'une stratégie d'attribution de noms de segment](#)

Ajout d'un filtre de suivi à votre application (Tomcat)

Pour Tomcat, ajoutez un `<filter>` aux fichier `web.xml` de votre projet. Utilisez le paramètre `fixedName` pour spécifier un [nom de service](#) à appliquer aux segments créés pour les demandes entrantes.

Exemple WEB-INF/web.xml - Tomcat

```
<filter>
  <filter-name>AWSXRayServletFilter</filter-name>
  <filter-class>com.amazonaws.xray.java.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>
```

Ajout d'un filtre de suivi à votre application (Spring)

Pour Spring, ajoutez un `Filter` à votre classe `WebConfig`. Transmettez le nom du segment au constructeur [AWSXRayServletFilter](#) en tant que chaîne.

Exemple src/main/java/myapp/WebConfig.java - printemps

```
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");
    }
}
```

Configuration d'une stratégie d'attribution de noms de segment

AWS X-Ray utilise un nom de service pour identifier votre application et la distinguer des autres applications, bases de données, ressources externes APIs et AWS ressources utilisées par votre application. Lorsque le SDK X-Ray génère des segments pour les demandes entrantes, il enregistre le nom du service de votre application dans le [champ du nom](#) du segment.

Le SDK X-Ray peut nommer les segments d'après le nom d'hôte dans l'en-tête de la requête HTTP. Cependant, cet en-tête peut être falsifié, ce qui peut entraîner la création de nœuds inattendus dans votre carte de service. Pour éviter que le SDK ne nomme les segments de manière incorrecte en raison de demandes contenant des en-têtes d'hôte falsifiés, vous devez spécifier un nom par défaut pour les demandes entrantes.

Si votre application traite des demandes pour plusieurs domaines, vous pouvez configurer le SDK pour qu'il utilise une stratégie de dénomination dynamique afin de refléter cela dans les noms des segments. Une stratégie de dénomination dynamique permet au SDK d'utiliser le nom d'hôte pour les demandes qui correspondent à un modèle attendu et d'appliquer le nom par défaut aux demandes qui ne le sont pas.

Par exemple, il se peut qu'une seule application envoie des demandes à trois sous-domaines : `www.example.com`, `api.example.com`, `etstatic.example.com`. Vous pouvez utiliser une stratégie de dénomination dynamique avec le modèle `*.example.com` pour identifier les segments de chaque sous-domaine avec un nom différent, ce qui permet d'obtenir trois nœuds de service sur

la carte des services. Si votre application reçoit des demandes dont le nom d'hôte ne correspond pas au modèle, vous verrez un quatrième nœud sur la carte des services avec un nom de remplacement que vous spécifiez.

Pour utiliser le même nom pour tous les segments de la demande, spécifiez le nom de votre application lorsque vous initialisez le filtre servlet, comme indiqué dans la [section précédente](#). Cela a le même effet que de créer un correctif [SegmentNamingStrategy](#) en l'appelant `SegmentNamingStrategy.fixed()` et en le transmettant au [AWSXRayServletFilter](#) constructeur.

Note

Vous pouvez remplacer le nom de service par défaut que vous définissez avec la `AWS_XRAY_TRACING_NAME` variable d'environnement [???](#).

Une stratégie d'attribution de noms dynamique définit un modèle auquel doivent correspondre les noms d'hôte et un nom par défaut à utiliser si le nom d'hôte de la demande HTTP ne correspond pas au modèle. Pour nommer les segments de façon dynamique dans Tomcat, utilisez `dynamicNamingRecognizedHosts` et `dynamicNamingFallbackName` pour définir, respectivement, le modèle et le nom par défaut.

Exemple WEB-INF/web.xml - Filtre servlet avec attribution de noms dynamique

```
<filter>
  <filter-name>AWSXRayServletFilter</filter-name>
  <filter-class>com.amazonaws.xray.java.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>
```

Pour Spring, créez une dynamique [SegmentNamingStrategy](#) en appelant `SegmentNamingStrategy.dynamic()` et transmettez-la au `AWSXRayServletFilter` constructeur.

Exemple `src/main/java/myapp/WebConfig.java` - filtre de servlet avec dénomination dynamique

```
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"));
    }
}
```

Suivi des appels du AWS SDK avec le SDK X-Ray pour Java

Lorsque votre application fait des appels Services AWS pour stocker des données, écrire dans une file d'attente ou envoyer des notifications, le SDK X-Ray pour Java suit les appels en aval [dans](#) des sous-segments. Les ressources tracées Services AWS et auxquelles vous accédez au sein de ces services (par exemple, un compartiment Amazon S3 ou une file d'attente Amazon SQS) apparaissent sous forme de nœuds en aval sur la carte de suivi de la console X-Ray.

Le SDK X-Ray pour Java instrumente automatiquement tous les clients du SDK AWS lorsque vous incluez `aws-sdk` les sous-modules et `aws-sdk-instrumentor` [an](#) dans votre build. Si vous n'incluez pas les sous-modules `Instrumentor`, vous pouvez choisir d'instrumenter certains clients tout en excluant d'autres.

Pour instrumenter des clients individuels, supprimez le `aws-sdk-instrumentor` sous-module de votre build et ajoutez-en un `XRayClient` en tant que `TracingHandler` sur votre client AWS SDK à l'aide du générateur de clients du service.

Par exemple, pour instrumenter un client AmazonDynamoDB, transmettez un gestionnaire de suivi à `AmazonDynamoDBClientBuilder`.

Exemple MyModel.java - Client DynamoDB

```
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();
    ...
}
```

Pour tous les services, vous pouvez voir le nom de l'API appelée dans la console X-Ray. Pour un sous-ensemble de services, le SDK X-Ray ajoute des informations au segment afin d'améliorer la granularité de la carte des services.

Par exemple, lorsque vous passez un appel avec un client DynamoDB instrumenté, le SDK ajoute le nom de la table au segment pour les appels qui ciblent une table. Dans la console, chaque table apparaît sous la forme d'un nœud distinct dans la carte des services, avec un nœud DynamoDB générique pour les appels qui ne ciblent pas une table.

Exemple Sous-segment pour un appel à DynamoDB pour enregistrer un élément

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

Lorsque vous accédez aux ressources nommées, les appels vers les services suivants créent des nœuds supplémentaires dans la cartographie des services. Les appels qui ne ciblent pas de ressources spécifiques créent un nœud générique pour le service.

- Amazon DynamoDB — Nom de la table
- Amazon Simple Storage Service : nom du compartiment et de la clé
- Amazon Simple Queue Service — Nom de la file d'attente

Pour instrumenter les appels en aval vers Services AWS AWS SDK pour Java 2.2 et versions ultérieures, vous pouvez omettre le `aws-xray-recorder-sdk-aws-sdk-v2-instrumentor` module de votre configuration de compilation. Incluez plutôt le module `aws-xray-recorder-sdk-aws-sdk-v2` module, puis instrumentez les clients individuels en les configurant avec un `TracingInterceptor`.

Exemple AWS SDK pour Java 2.2 et versions ultérieures - intercepteur de traçage

```
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();  
    //...  
}
```

Suivi des appels vers les services Web HTTP en aval avec le SDK X-Ray pour Java

Lorsque votre application effectue des appels vers des microservices ou du protocole HTTP public APIs, vous pouvez utiliser la version de X-Ray SDK pour Java pour `HttpClient` instrumenter ces appels et ajouter l'API au graphe de service en tant que service en aval.

Le SDK X-Ray pour Java `DefaultHttpClient` inclut des `HttpClientBuilder` classes qui peuvent être utilisées à la place des équivalents `HttpComponents` Apache pour instrumenter les appels HTTP sortants.

- `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`

Ces bibliothèques se trouvent dans le sous-module [aws-xray-recorder-sdk-apache-http](#).

Vous pouvez remplacer vos relevés d'importation existants par l'équivalent de X-Ray pour instrumenter tous les clients, ou utiliser le nom complet lorsque vous initialisez un client pour instrumenter des clients spécifiques.

Exemple `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();
}
}
```

Lorsque vous instrumentez un appel à une API Web en aval, le SDK X-Ray pour Java enregistre un sous-segment contenant des informations sur la requête et la réponse HTTP. X-Ray utilise le sous-segment pour générer un segment inféré pour l'API distante.

Exemple Sous-segment pour un appel HTTP en aval

```
{
  "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
    }
  }
}
```

Exemple Segment déduit pour un appel HTTP en aval

```
{
  "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
}
```

Suivi des requêtes SQL avec le SDK X-Ray pour Java

Intercepteurs SQL

Instrumentez les requêtes de base de données SQL en ajoutant l'intercepteur JDBC du SDK X-Ray pour Java à la configuration de votre source de données.

- PostgreSQL – `com.amazonaws.xray.sql.postgres.TracingInterceptor`
- MySQL – `com.amazonaws.xray.sql.mysql.TracingInterceptor`

Ces intercepteurs se trouvent respectivement dans les sous-modules [aws-xray-recorder-sql-postgres](#) et [aws-xray-recorder-sql-mysql](#). Ils implémentent `org.apache.tomcat.jdbc.pool.JdbcInterceptor` et sont compatibles avec les groupes de connexion Tomcat.

Note

Les intercepteurs SQL n'enregistrent pas la requête SQL elle-même dans les sous-segments à des fins de sécurité.

Pour Spring, ajoutez l'intercepteur dans un fichier de propriétés et créez la source de données avec le `DataSourceBuilder` de Spring Boot.

Exemple **`src/main/java/resources/application.properties`** - Intercepteur JDBC PostgreSQL

```
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
```

Exemple `src/main/java/myapp/WebConfig.java` - Source de données

```
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();
    }
    ...
}
```

Pour Tomcat, faites appel à `setJdbcInterceptors` la source de données JDBC en faisant référence à la classe X-Ray SDK for Java.

Exemple `src/main/myapp/model.java` - Source de données

```
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");
```

La bibliothèque de sources de données Tomcat JDBC est incluse dans le SDK X-Ray pour Java, mais vous pouvez la déclarer comme dépendance fournie pour documenter que vous l'utilisez.

Exemple `pom.xml` - Source de données JDBC

```
<dependency>
  <groupId>org.apache.tomcat</groupId>
  <artifactId>tomcat-jdbc</artifactId>
  <version>8.0.36</version>
  <scope>provided</scope>
</dependency>
```

Décorateur de traçage SQL natif

- Ajoutez [aws-xray-recorder-sdk-sql](#) à vos dépendances.
- Décorez la source de données, la connexion ou l'instruction de votre base de données.

```
dataSource = TracingDataSource.decorate(dataSource)
connection = TracingConnection.decorate(connection)
statement = TracingStatement.decorateStatement(statement)
preparedStatement = TracingStatement.decoratePreparedStatement(preparedStatement,
    sql)
callableStatement = TracingStatement.decorateCallableStatement(callableStatement,
    sql)
```

Génération de sous-segments personnalisés avec le SDK X-Ray pour Java

Les sous-segments étendent le [segment](#) d'une trace avec des détails sur le travail effectué afin de répondre à une demande. Chaque fois que vous passez un appel avec un client instrumenté, le SDK

X-Ray enregistre les informations générées dans un sous-segment. Vous pouvez créer des sous-segments supplémentaires pour regrouper d'autres sous-segments, pour mesurer les performances d'une section de code ou pour enregistrer des annotations et des métadonnées.

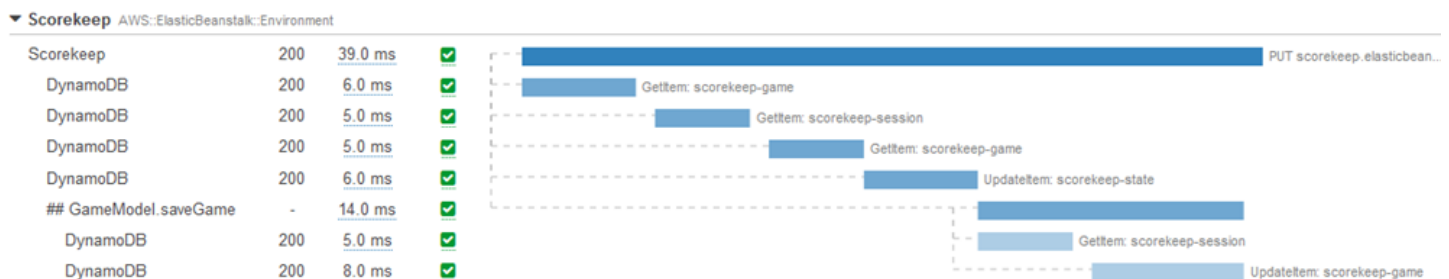
Pour gérer des sous-segments, utilisez les méthodes `beginSubsegment` et `endSubsegment`.

Exemple `GameModel.java` - sous-segment personnalisé

```
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();
    }
}
```

Dans cet exemple, le code du sous-segment charge la session du jeu depuis DynamoDB à l'aide d'une méthode basée sur le modèle de session et utilise le mappeur DynamoDB pour enregistrer le AWS SDK pour Java jeu. L'encapsulation de ce code dans un sous-segment fait des appels DynamoDB des enfants du sous-segment dans `Save Game` la vue de suivi de la console.



Si le code de votre sous-segment lève des exceptions cochées, enveloppez-le dans un bloc `try` et appelez `AWSXRay.endSubsegment()` dans un bloc `finally` pour veiller à ce que le sous-segment

soit toujours fermé. Si un sous-segment n'est pas fermé, le segment parent ne peut pas être terminé et ne sera pas envoyé à X-Ray.

Pour le code qui ne génère pas d'exceptions vérifiées, vous pouvez le transmettre `AWSXRay.CreateSubsegment` en tant que fonction Lambda.

Exemple Sous-segment de la fonction Lambda

```
import com.amazonaws.xray.AWSXRay;

AWSXRay.createSubsegment("getMovies", (subsegment) -> {
    // function code
});
```

Lorsque vous créez un sous-segment au sein d'un segment ou d'un autre sous-segment, le SDK X-Ray pour Java génère un identifiant et enregistre l'heure de début et de fin.

Exemple Sous-segment avec des métadonnées

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

Pour la programmation asynchrone et multithread, vous devez transmettre manuellement le sous-segment à la `endSubsegment()` méthode pour vous assurer qu'il est correctement fermé, car le contexte de X-Ray peut être modifié lors de l'exécution asynchrone. Si un sous-segment asynchrone est fermé après la fermeture de son segment parent, cette méthode diffuse automatiquement le segment entier vers le daemon X-Ray.

Exemple Sous-segment asynchrone

```
@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();
}
```

Ajoutez des annotations et des métadonnées aux segments avec le SDK X-Ray pour Java

Vous pouvez enregistrer des informations supplémentaires sur les demandes, l'environnement ou votre application à l'aide d'annotations et de métadonnées. Vous pouvez ajouter des annotations et des métadonnées aux segments créés par le SDK X-Ray ou aux sous-segments personnalisés que vous créez.

Les annotations sont des paires clé-valeur avec des chaînes, des nombres ou des valeurs booléennes. Les annotations sont indexées pour être utilisées avec les expressions de [filtre](#). Utilisez les annotations pour enregistrer les données que vous souhaitez utiliser pour regrouper les suivis dans la console ou lors de l'appel de l'API [GetTraceSummaries](#).

Les métadonnées sont des paires clé-valeur qui peuvent contenir des valeurs de n'importe quel type, y compris des objets et des listes, mais qui ne sont pas indexées pour être utilisées avec des expressions de filtre. Utilisez les métadonnées pour enregistrer des données supplémentaires que vous souhaitez stocker dans le traçage, mais que vous n'avez pas besoin d'utiliser pour la recherche.

En plus des annotations et des métadonnées, vous pouvez également [enregistrer les chaînes d'ID utilisateur](#) sur des segments. Les IDs Les utilisateurs sont enregistrés dans un champ distinct sur les segments et sont indexés pour être utilisés dans le cadre de la recherche.

Sections

- [Enregistrer des annotations avec le SDK X-Ray pour Java](#)
- [Enregistrement de métadonnées avec le SDK X-Ray pour Java](#)
- [Utilisateur enregistrant IDs avec le SDK X-Ray pour Java](#)

Enregistrer des annotations avec le SDK X-Ray pour Java

Utilisez les annotations pour enregistrer les informations sur les segments ou sous-segments qui doivent être indexés pour la recherche.

Exigences liées aux annotations

- Clés — La clé d'une annotation X-Ray peut comporter jusqu'à 500 caractères alphanumériques. Vous ne pouvez pas utiliser d'espaces ou de symboles autres qu'un point ou un point (.)
- Valeurs — La valeur d'une annotation X-Ray peut comporter jusqu'à 1 000 caractères Unicode.
- Nombre d'annotations : vous pouvez utiliser jusqu'à 50 annotations par trace.

Pour enregistrer les annotations

1. Obtenez une référence au segment ou sous-segment en cours à partir d'AWSXRay.

```
import com.amazonaws.xray.AWSXRay;  
import com.amazonaws.xray.entities.Segment;  
...  
Segment document = AWSXRay.getCurrentSegment();
```

or

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

2. Appelez `putAnnotation` avec une clé de type chaîne et une valeur de type booléen, numérique ou chaîne.

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

L'exemple suivant montre comment appeler `putAnnotation` avec une clé `String` qui inclut un point et une valeur booléenne, numérique ou chaîne.

```
document.putAnnotation("testkey.test", "my value");
```

Le kit de développement logiciel enregistre les annotations sous forme de paires clé-valeur dans un objet annotations du document de segment. Si vous appelez deux fois `putAnnotation` avec la même clé, les valeurs précédemment enregistrées sur le même segment ou sous-segment sont remplacées.

Pour rechercher les suivis ayant des annotations avec des valeurs spécifiques, utilisez le mot clé `annotation[key]` dans une [expression de filtre](#).

Exemple [src/main/java/scorekeep/GameModel1.java](#)— Annotations et métadonnées

```
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("## GameModel1.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();
    }
}
```

Enregistrement de métadonnées avec le SDK X-Ray pour Java

Utilisez les métadonnées pour enregistrer des informations sur les segments ou sous-segments qui n'ont pas besoin d'être indexés pour la recherche. Les valeurs des métadonnées peuvent être de type chaîne, valeur numérique, valeur booléenne ou tout objet à même d'être sérialisé en un tableau ou objet JSON.

Pour enregistrer les métadonnées

1. Obtenez une référence au segment ou sous-segment en cours à partir d'AWSXRay.

```
import com.amazonaws.xray.AWSXRay;
import com.amazonaws.xray.entities.Segment;
...
Segment document = AWSXRay.getCurrentSegment();
```

or

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

2. Appelez `putMetadata` avec un espace de noms de type chaîne, une clé de type chaîne, et une valeur de type booléen, numérique, chaîne ou objet.

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

or

Appelez `putMetadata` avec seulement une clé et une valeur.

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

Si vous ne spécifiez pas d'espace de noms, le kit de développement logiciel utilise `default`. Si vous appelez deux fois `putMetadata` avec la même clé, les valeurs précédemment enregistrées sur le même segment ou sous-segment sont remplacées.

Exemple [src/main/java/scorekeep/GameModel.java](#)— Annotations et métadonnées

```
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();
}
}
```

Utilisateur enregistrant IDs avec le SDK X-Ray pour Java

Enregistrez les segments utilisateur IDs sur demande pour identifier l'utilisateur qui a envoyé la demande.

Pour enregistrer un utilisateur IDs

1. Obtenez une référence au segment en cours à partir d'AWSXRay.

```
import com.amazonaws.xray.AWSXRay;
import com.amazonaws.xray.entities.Segment;
...
Segment document = AWSXRay.getCurrentSegment();
```

2. Appelez setUser avec l'ID de type chaîne de l'utilisateur ayant envoyé la demande.

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

Vous pouvez appeler setUser dans vos contrôleurs pour enregistrer l'ID utilisateur dès que votre application commence le traitement d'une demande. Si vous n'utilisez le segment que pour définir l'ID utilisateur, vous pouvez chaîner les appels sur une seule ligne.

Exemple [src/main/java/scorekeep/MoveController.java](#) — ID utilisateur

```
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);
}
```

Pour rechercher les suivis d'un ID utilisateur, utilisez le mot clé user dans une [expression de filtre](#).

AWS X-Ray métriques pour le SDK X-Ray pour Java

Cette rubrique décrit l'espace de AWS X-Ray noms, les métriques et les dimensions. Vous pouvez utiliser le SDK X-Ray pour Java pour publier des métriques CloudWatch Amazon non échantillonnées à partir des segments X-Ray que vous avez collectés. Ces métriques sont dérivées de l'heure de début et de fin du segment, ainsi que des indicateurs d'erreur, de défaillance et d'état de limitation. Utilisez ces métriques de suivi pour exposer les nouvelles tentatives et les problèmes de dépendance dans les sous-segments.

CloudWatch est un référentiel de métriques. Une métrique est le concept fondamental CloudWatch et représente un ensemble de points de données triés dans le temps. Vous publiez (ou Services AWS) publiez des points de données métriques dans CloudWatch lesquels vous extrayez des statistiques sous la forme d'un ensemble ordonné de séries chronologiques.

Les métriques sont uniquement définies par un nom, un espace de noms et une ou plusieurs dimensions. Chaque point de données comporte un horodatage et, le cas échéant, une unité de mesure. Lorsque vous demandez des statistiques, le flux de données renvoyé est identifié par l'espace de noms, le nom de la métrique et la dimension.

Pour plus d'informations CloudWatch, consultez le [guide de CloudWatch l'utilisateur Amazon](#).

CloudWatch Métriques X-Ray

L'espace de noms ServiceMetrics/SDK inclut les métriques suivantes.

Métrique	Statistiques disponibles	Description	Unités
Latency	Moyenne, Minimum, Maximum, Nombre	Différence entre l'heure de début et de fin. Moyenne, minimale et maximale décrivent tous une latence de fonctionnement. Count décrit le nombre d'appels.	Millisecondes
ErrorRate	Average, Sum	Taux de demandes qui ont échoué avec un code d'état 4xx Client Error, entraînant une erreur.	Pourcentage
FaultRate	Average, Sum	Taux de traces qui ont échoué avec un code d'état 5xx Server Error, entraînant une erreur.	Pourcentage
ThrottleRate	Average, Sum	Le taux de traces limitées qui renvoient un code d'état 429. Il s'agit d'un sous-ensemble de la métrique ErrorRate .	Pourcentage
OkRate	Average, Sum	Taux de demandes suivies entraînant un code d'état OK.	Pourcentage

CloudWatch Dimensions du X-Ray

Utilisez les dimensions du tableau suivant pour affiner les mesures renvoyées pour votre instrument X-Ray Java applications.

Dimension	Description
ServiceType	Type du service, par exemple, <code>AWS::EC2::Instance</code> ou <code>NONE</code> , s'il n'est pas connu.
ServiceName	Nom canonique du service.

Activez les CloudWatch métriques X-Ray

Utilisez la procédure suivante pour activer les mesures de suivi dans votre instrument Java application.

Pour configurer les métriques de suivi

1. Ajoutez le `aws-xray-recorder-sdk-metrics` package en tant que Apache Maven dépendance. Pour plus d'informations, consultez la section Sous-modules du [SDK X-Ray pour Java](#).
2. Activez un nouveau dans le `MetricsSegmentListener()` comme partie intégrante de la construction de l'enregistreur global.

Exemple `src/com/myapp/web/Startup.java`

```
import com.amazonaws.xray.AWSXRay;
import com.amazonaws.xray.AWSXRayRecorderBuilder;
import com.amazonaws.xray.plugins.EC2Plugin;
import com.amazonaws.xray.plugins.ElasticBeanstalkPlugin;
import com.amazonaws.xray.strategy.sampling.LocalizedSamplingStrategy;

@Configuration
public class WebConfig {
    ...
    static {
        AWSXRayRecorderBuilder builder = AWSXRayRecorderBuilder
            .standard()
            .withPlugin(new EC2Plugin())
```

```
        .withPlugin(new ElasticBeanstalkPlugin())
        .withSegmentListener(new
MetricsSegmentListener());

    URL ruleFile = WebConfig.class.getResource("/sampling-rules.json");
    builder.withSamplingStrategy(new LocalizedSamplingStrategy(ruleFile));

    AWSXRay.setGlobalRecorder(builder.build());
}
}
```

3. Déployez l' CloudWatch agent pour collecter des métriques à l'aide d'Amazon Elastic Compute Cloud (Amazon EC2), d'Amazon Elastic Container Service (Amazon ECS) ou d'Amazon Elastic Kubernetes Service (Amazon EKS) :
 - Pour configurer Amazon EC2, consultez la section [Installation de l' CloudWatch agent](#).
 - Pour configurer Amazon ECS, consultez [Surveiller les conteneurs Amazon ECS à l'aide de Container Insights](#).
 - Pour configurer Amazon EKS, consultez [Installer l' CloudWatch agent à l'aide du module complémentaire Amazon CloudWatch Observability EKS](#).
4. Configurez le SDK pour communiquer avec l' CloudWatch agent. Par défaut, le SDK communique avec l' CloudWatch agent sur l'adresse 127.0.0.1. Vous pouvez configurer d'autres adresses en définissant la variable d'environnement ou la propriété Java sur address:port.

Exemple Variable d'environnement

```
AWS_XRAY_METRICS_DAEMON_ADDRESS=address:port
```

Exemple Propriété Java

```
com.amazonaws.xray.metrics.daemonAddress=address:port
```

Pour valider la configuration

1. Connectez-vous à la CloudWatch console AWS Management Console et ouvrez-la à l'adresse <https://console.aws.amazon.com/cloudwatch/>.

2. Ouvrez l'onglet Métriques pour observer le flot de vos métriques.
3. (Facultatif) Dans la CloudWatch console, sous l'onglet Journaux, ouvrez le groupe de ServiceMetricsSDK journaux. Recherchez un flux de journal correspondant aux métriques de l'hôte et confirmez les messages de journal.

Transmission de contexte de segment entre threads dans une application multithreads

Lorsque vous créez un thread dans l'application, `AWSXRayRecorder` ne conserve pas de référence au segment ou sous-segment actuel [Entity](#). Si vous utilisez un client instrumenté dans le nouveau thread, le SDK essaie d'écrire sur un segment qui n'existe pas, ce qui provoque un [SegmentNotFoundException](#).

Pour éviter de générer des exceptions pendant le développement, vous pouvez configurer l'enregistreur avec un [ContextMissingStrategy](#) qui lui indique de consigner une erreur à la place. Vous pouvez configurer la stratégie en code avec [SetContextMissingStrategy](#), une variable d'environnement ou une propriété système ou configurer des options équivalentes avec une [variable d'environnement](#) ou une [propriété système](#).

Pour résoudre l'erreur, vous pouvez utiliser un nouveau segment en appelant [beginSegment](#) lorsque vous démarrez le thread et [endSegment](#) lorsque vous le fermez. Cela fonctionne si vous instrumentez du code qui ne s'exécute pas en réponse à une demande HTTP, à l'image du code qui s'exécute au démarrage de l'application.

Si vous utilisez plusieurs threads pour gérer les demandes entrantes, vous pouvez transmettre le segment ou le sous-segment en cours au nouveau thread et le fournir à l'enregistreur mondial. Cela permet de s'assurer que les informations enregistrées dans le nouveau thread sont associées au même segment que les autres informations enregistrées concernant cette demande. Une fois que le segment est disponible dans le nouveau thread, vous pouvez exécuter n'importe quel exécutable ayant accès au contexte de ce segment à l'aide de la `segment.run(() -> { ... })` méthode.

Veuillez consulter [Utilisation de clients instrumentés dans les threads de travail](#) pour obtenir un exemple.

Utilisation de X-Ray avec une programmation asynchrone

Le SDK X-Ray pour Java peut être utilisé dans des programmes Java asynchrones avec. [SegmentContextExecutors](#) `SegmentContextExecutor` implémente l'interface `Executor`, ce qui signifie

qu'elle peut être transmise à toutes les opérations asynchrones d'un. [CompletableFuture](#) Cela garantit que toutes les opérations asynchrones seront exécutées avec le segment approprié dans son contexte.

Exemple Exemple App.java : Passage SegmentContextExecutor à 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());
```

AOP avec Spring et le SDK X-Ray pour Java

Cette rubrique décrit comment utiliser le SDK X-Ray et le Spring Framework pour instrumenter votre application sans en modifier la logique de base. Cela signifie qu'il existe désormais un moyen non invasif d'instrumenter vos applications exécutées à AWS distance.

Pour activer AOP dans Spring

1. [Configurez Spring](#)
2. [Ajoutez un filtre de suivi à votre application](#)
3. [Annotez votre code ou implémentez une interface](#)
4. [Activez X-Ray dans votre application](#)

Configuration de Spring

Vous pouvez utiliser Maven ou Gradle pour configurer Spring afin de vous servir d'AOP pour instrumenter votre application.

Si vous utilisez Maven pour créer votre application, ajoutez la dépendance suivante dans votre fichier pom.xml.

```
<dependency>
```

```
<groupId>com.amazonaws</groupId>
<artifactId>aws-xray-recorder-sdk-spring</artifactId>
<version>2.11.0</version>
</dependency>
```

Pour Gradle, ajoutez la dépendance suivante dans votre fichier `build.gradle`.

```
compile 'com.amazonaws:aws-xray-recorder-sdk-spring:2.11.0'
```

Configuration de Spring Boot

Outre la dépendance Spring décrite dans la section précédente, si vous utilisez Spring Boot, ajoutez la dépendance suivante si elle ne figure pas déjà dans votre chemin de classe.

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'
```

Ajouter un filtre de suivi à votre application

Ajoutez un `Filter` à votre `WebConfig` classe. Transmettez le nom du segment au constructeur [AWSXRayServletFilter](#) en tant que chaîne. Pour plus d'informations sur le suivi des filtres et l'instrumentation des demandes entrantes, consultez [Suivi des demandes entrantes avec le SDK X-Ray pour Java](#).

Exemple `src/main/java/myapp/WebConfig.java` - printemps

```
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");
    }
}
```

Support à Jakarta

Spring 6 utilise [Jakarta](#) au lieu de Javax pour son édition Enterprise. Pour prendre en charge ce nouvel espace de noms, X-Ray a créé un ensemble parallèle de classes qui vivent dans leur propre espace de noms Jakarta.

Pour les classes de filtres, remplacez javax par jakarta. Lorsque vous configurez une stratégie de dénomination de segment, ajoutez-la jakarta avant le nom de classe de stratégie de dénomination, comme dans l'exemple suivant :

```
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"));
    }
}
```

Annotation de votre code ou implémentation d'une interface

Vos classes doivent soit être annotées avec l'`@XRayEnabled` annotation, soit implémenter l'`XRayTraced` interface. Cela demande au système AOP d'encapsuler les fonctions de la classe affectée pour l'instrumentation X-Ray.

Activer X-Ray dans votre application

Pour activer le traçage X-Ray dans votre application, votre code doit étendre la classe abstraite `BaseAbstractXRayInterceptor` en remplaçant les méthodes suivantes.

- `generateMetadata`—Cette fonction permet de personnaliser les métadonnées associées à la trace de la fonction en cours. Par défaut, le nom de classe de la fonction d'exécution est enregistré dans les métadonnées. Vous pouvez ajouter des données supplémentaires si vous avez besoin d'informations supplémentaires.
- `xrayEnabledClasses`—Cette fonction est vide et doit le rester. Elle sert d'hôte pour un pointcut demandant à l'intercepteur quelles méthodes encapsuler. Définissez le pointcut en spécifiant les classes qui sont annotées avec `@XRayEnabled` pour effectuer le suivi. L'instruction de pointcut suivante demande à l'intercepteur d'encapsuler tous les beans de contrôleur annotés avec l'annotation `@XRayEnabled`.

```
@Pointcut("@within(com.amazonaws.xray.spring.aop.XRayEnabled) && bean(*Controller)")
```

Si votre projet utilise Spring Data JPA, envisagez d'étendre depuis `AbstractXRayInterceptor` au lieu de `BaseAbstractXRayInterceptor`.

exemple

Le code suivant étend la classe abstraite `BaseAbstractXRayInterceptor`.

```
@Aspect
@Component
public class XRayInspector extends BaseAbstractXRayInterceptor {
    @Override
    protected Map<String, Map<String, Object>> generateMetadata(ProceedingJoinPoint
        proceedingJoinPoint, Subsegment subsegment) throws Exception {
        return super.generateMetadata(proceedingJoinPoint, subsegment);
    }

    @Override
    @Pointcut("@within(com.amazonaws.xray.spring.aop.XRayEnabled) && bean(*Controller)")

    public void xrayEnabledClasses() {}
}
```

Le code suivant est une classe qui sera instrumentée par X-Ray.

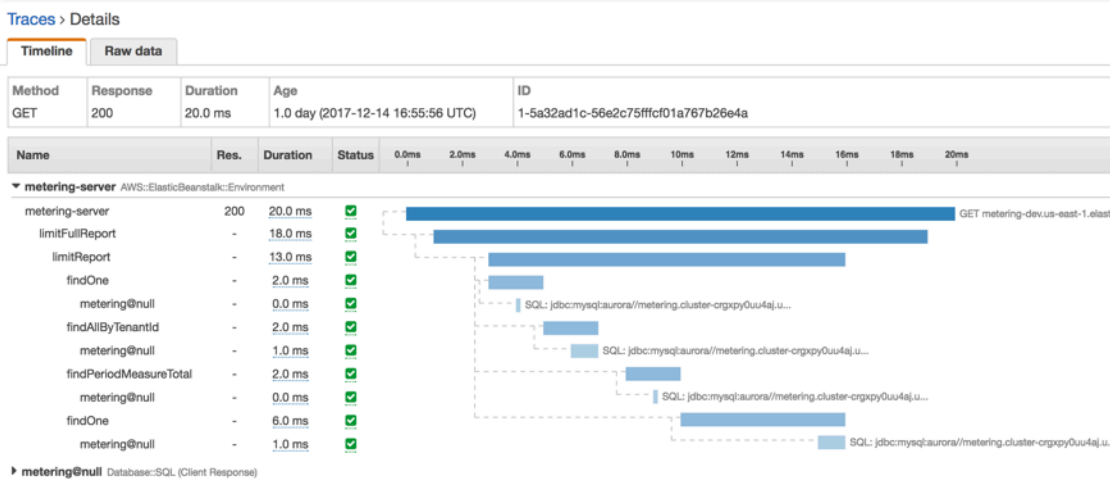
```
@Service
@XRayEnabled
public class MyServiceImpl implements MyService {
    private final MyEntityRepository myEntityRepository;

    @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());
        }
    }
}
```

Si vous avez correctement configuré votre application, vous devez voir la pile d'appels complète de l'application, depuis le contrôleur jusqu'aux appels de service, comme illustré dans la capture d'écran suivante de la console.



Utilisation de Node.js

Il existe deux manières d'instrumenter votre application Node.js pour envoyer des traces à X-Ray :

- [AWS Distro for OpenTelemetry JavaScript](#) : AWS distribution qui fournit un ensemble de bibliothèques open source permettant d'envoyer des métriques et des traces corrélées à plusieurs solutions de AWS surveillance, notamment Amazon et Amazon OpenSearch Service CloudWatch AWS X-Ray, via [AWS Distro](#) for Collector. OpenTelemetry
- [AWS X-Ray SDK pour Node.js](#) : ensemble de bibliothèques permettant de générer et d'envoyer des traces à X-Ray via le [daemon X-Ray](#).

Pour de plus amples informations, veuillez consulter [Choisir entre The AWS Distro for OpenTelemetry et X-Ray SDKs](#).

AWS Distro pour OpenTelemetry JavaScript

Avec le AWS Distro for OpenTelemetry (ADOT) JavaScript, vous pouvez instrumenter vos applications une seule fois et envoyer des métriques et des traces corrélées à plusieurs solutions de AWS surveillance CloudWatch, notamment Amazon AWS X-Ray et Amazon Service. OpenSearch L'utilisation de X-Ray avec AWS Distro pour OpenTelemetry nécessite deux composants : un OpenTelemetry SDK activé pour une utilisation avec X-Ray, et un AWS Distro for OpenTelemetry Collector activé pour une utilisation avec X-Ray.

Pour commencer, consultez la [OpenTelemetry JavaScript documentation de la AWS distribution](#).

Note

ADOT JavaScript est pris en charge pour toutes les applications Node.js côté serveur. ADOT n' JavaScript est pas en mesure d'exporter des données vers X-Ray à partir de clients de navigateur.

Pour plus d'informations sur l'utilisation du AWS Distro pour OpenTelemetry with AWS X-Ray et d'autres Services AWS, voir [AWS Distro for OpenTelemetry](#) ou [AWS Distro pour la documentation](#). OpenTelemetry

Pour plus d'informations sur la prise en charge et l'utilisation des langues, consultez [AWS Observability on GitHub](#).

AWS Kit de développement X-Ray pour Node.js

Le SDK X-Ray pour Node.js est une bibliothèque pour les applications Web Express et les fonctions Lambda Node.js qui fournit des classes et des méthodes permettant de générer et d'envoyer des données de trace au démon X-Ray. Les données de suivi incluent des informations sur les requêtes HTTP entrantes traitées par l'application et les appels que l'application effectue aux services en aval à l'aide du AWS SDK ou des clients HTTP.

Note

Le SDK X-Ray pour Node.js est un projet open source pris en charge pour les versions 14.x et supérieures de Node.js. Vous pouvez suivre le projet et soumettre des problèmes et des pull requests sur GitHub : github.com/aws/aws-xray-sdk-node

Si vous utilisez Express, commencez par [ajouter le SDK comme intergiciel](#) sur votre serveur d'applications afin de suivre les demandes entrantes. L'intergiciel crée un [segment](#) pour chaque demande suivie et termine le segment lorsque la réponse est envoyée. Tandis que le segment soit ouvert, vous pouvez utiliser les méthodes du client du SDK pour ajouter des informations au segment et créer des sous-segments afin de suivre les appels en amont. Le kit de développement logiciel (SDK) enregistre aussi automatiquement les exceptions que votre application renvoie quand le segment est ouvert.

Pour les fonctions Lambda appelées par une application ou un service instrumenté, Lambda lit l'[en-tête de suivi et trace automatiquement les requêtes échantillonnées](#). Pour les autres fonctions, vous pouvez [configurer Lambda](#) pour échantillonner et suivre les demandes entrantes. Dans les deux cas, Lambda crée le segment et le fournit au SDK X-Ray.

Note

Sur Lambda, le SDK X-Ray est facultatif. Si vous ne l'utilisez pas dans votre fonction, votre carte de service inclura toujours un nœud pour le service Lambda et un nœud pour chaque fonction Lambda. En ajoutant le SDK, vous pouvez instrumenter votre code de fonction pour ajouter des sous-segments au segment de fonction enregistré par Lambda. Pour plus d'informations, consultez [AWS Lambda et AWS X-Ray](#).

Utilisez ensuite le SDK X-Ray pour Node.js afin d'[instrumenter votre AWS SDK pour JavaScript les clients Node.js](#). Chaque fois que vous appelez une ressource Service AWS ou une ressource en aval avec un client instrumenté, le SDK enregistre les informations relatives à l'appel dans un sous-segment. Services AWS et les ressources auxquelles vous accédez au sein des services apparaissent sous forme de nœuds en aval sur la carte de trace pour vous aider à identifier les erreurs et les problèmes de limitation sur les connexions individuelles.

Le kit de développement X-Ray pour Node.js fournit également des instruments pour les appels en aval aux requêtes Web APIs et SQL HTTP. [Enveloppez votre client HTTP dans la méthode de capture du kit de développement logiciel \(SDK\)](#) afin d'enregistrer les informations sur les appels HTTP sortants. Pour les clients SQL, [utilisez la méthode de capture correspondant à votre type de base de données](#).

L'intergiciel applique les règles d'échantillonnage aux demandes entrantes pour déterminer les demandes à suivre. Vous pouvez [configurer le SDK X-Ray pour Node.js](#) afin d'ajuster le comportement d'échantillonnage ou d'enregistrer des informations concernant les ressources de AWS calcul sur lesquelles votre application s'exécute.

Enregistrez les informations supplémentaires sur les demandes et le travail que votre application effectue dans les [annotations et les métadonnées](#). Les annotations sont de simples paires clé-valeur, indexées en vue de leur utilisation avec les [expressions de filtre](#), de telle sorte que vous pouvez explorer les suivis qui contiennent des données spécifiques. Les entrées des métadonnées sont moins restrictives et peuvent enregistrer des objets et tableaux entiers (tout ce qui peut être sérialisé en JSON).

Annotations et métadonnées

Les annotations et les métadonnées sont du texte arbitraire que vous ajoutez aux segments avec le kit de développement X-Ray. Les annotations sont indexées pour être utilisées avec les expressions de filtre. Les métadonnées ne sont pas indexées, mais peuvent être consultées dans le segment brut à l'aide de la console ou de l'API X-Ray. Toute personne à qui vous accordez un accès en lecture à X-Ray peut consulter ces données.

Lorsque vous avez un grand nombre de clients instrumentés dans votre code, un seul segment de demande peut contenir un grand nombre de sous-segments, un par appel effectué à l'aide d'un client instrumenté. Vous pouvez organiser et grouper les sous-segments en enveloppant les appels clients dans des [sous-segments personnalisés](#). Vous pouvez créer un sous-segment personnalisé pour

une fonction complète ou une quelconque section du code, puis enregistrer les métadonnées et les annotations sur le sous-segment au lieu de tout écrire sur le segment parent.

Pour obtenir de la documentation de référence sur les classes et les méthodes du SDK, consultez le manuel de référence de l'[API du AWS X-Ray SDK pour Node.js](#).

Prérequis

Le SDK X-Ray pour Node.js nécessite Node.js et les bibliothèques suivantes :

- `atomic-batcher`— 1,0,2
- `cls-hooked`— 4.2.2
- `pkginfo`— 0,4,0
- `semver`— 5.3.0

Le kit SDK extrait les bibliothèques lorsque vous l'installez avec NPM.

Pour suivre les clients du AWS SDK, le SDK X-Ray pour Node.js nécessite une version minimale du AWS SDK pour JavaScript Node.js.

- `aws-sdk`— 2,7,15

Gestion des dépendances

Le SDK X-Ray pour Node.js est disponible auprès de NPM.

- Package — [aws-xray-sdk](#)

Pour le développement local, installez le kit de développement logiciel (SDK) dans le répertoire de votre projet avec npm.

```
~/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
```

Utilisez l'option `--save` pour sauvegarder le SDK comme dépendance dans le package `.json` de votre application.

```
~/nodejs-xray$ npm install aws-xray-sdk --save
aws-xray-sdk@3.3.3
```

Si votre application possède des dépendances dont les versions entrent en conflit avec les dépendances du SDK X-Ray, les deux versions seront installées pour garantir la compatibilité. Pour plus de détails, consultez la [documentation officielle de NPM pour la résolution des dépendances](#).

Exemples Node.js

Utilisez le AWS X-Ray SDK pour Node.js pour avoir une end-to-end vue d'ensemble des demandes lorsqu'elles transitent par vos applications Node.js.

- [Exemple d'application Node.js](#) sur GitHub.

Configuration du SDK X-Ray pour Node.js

Vous pouvez configurer le SDK X-Ray pour Node.js à l'aide de plug-ins afin d'inclure des informations sur le service sur lequel votre application s'exécute, de modifier le comportement d'échantillonnage par défaut ou d'ajouter des règles d'échantillonnage qui s'appliquent aux demandes adressées à des chemins spécifiques.

Sections

- [Plug-ins de service](#)
- [Règles d'échantillonnage](#)
- [Journalisation](#)
- [Adresse du daemon X-Ray](#)

- [Variables d'environnement](#)

Plug-ins de service

Permet plug-ins d'enregistrer des informations sur le service hébergeant votre application.

Plugins

- Amazon EC2 — EC2Plugin ajoute l'ID de l'instance, la zone de disponibilité et le groupe de CloudWatch journaux.
- Elastic ElasticBeanstalkPlugin Beanstalk : ajoute le nom de l'environnement, l'étiquette de version et l'ID de déploiement.
- Amazon ECS — ECSPlugin ajoute l'ID du conteneur.

Pour utiliser un plugin, configurez le SDK X-Ray pour le client Node.js à l'aide de la `config` méthode.

Exemple app.js - plug-ins

```
var AWSXRay = require('aws-xray-sdk');
AWSXRay.config([AWSXRay.plugins.EC2Plugin, AWSXRay.plugins.ElasticBeanstalkPlugin]);
```

Le SDK utilise également les paramètres du plugin pour définir le `origin` champ du segment. Cela indique le type de AWS ressource qui exécute votre application. Lorsque vous utilisez plusieurs plug-ins, le SDK utilise l'ordre de résolution suivant pour déterminer l'origine : ElasticBeanstalk > EKS > ECS > EC2.

Règles d'échantillonnage

Le SDK utilise les règles d'échantillonnage que vous définissez dans la console X-Ray pour déterminer les demandes à enregistrer. La règle par défaut suit la première demande chaque seconde, et 5 % de toutes les demandes supplémentaires provenant de tous les services envoient des traces à X-Ray. [Créez des règles supplémentaires dans la console X-Ray](#) pour personnaliser la quantité de données enregistrées pour chacune de vos applications.

Le SDK applique les règles personnalisées dans l'ordre dans lequel elles sont définies. Si une demande correspond à plusieurs règles personnalisées, le SDK applique uniquement la première règle.

 Note

Si le SDK ne parvient pas à accéder à X-Ray pour obtenir des règles d'échantillonnage, il revient à une règle locale par défaut concernant la première demande chaque seconde, et 5 % des demandes supplémentaires par hôte. Cela peut se produire si l'hôte n'est pas autorisé à appeler sampling APIs ou ne peut pas se connecter au daemon X-Ray, qui agit comme un proxy TCP pour les appels d'API effectués par le SDK.

Vous pouvez également configurer le SDK pour charger des règles d'échantillonnage à partir d'un document JSON. Le SDK peut utiliser les règles locales comme solution de rechange dans les cas où l'échantillonnage X-Ray n'est pas disponible, ou utiliser exclusivement les règles locales.

Exemple 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
  }
}
```

Cet exemple définit une règle personnalisée et une règle par défaut. La règle personnalisée applique un taux d'échantillonnage de 5 % sans nombre minimum de demandes à suivre pour les chemins sous-jacents. `/api/move/` La règle par défaut suit la première demande chaque seconde et 10 % des demandes supplémentaires.

L'inconvénient de définir des règles localement est que la cible fixe est appliquée par chaque instance de l'enregistreur indépendamment, au lieu d'être gérée par le service X-Ray. Au fur et à

mesure que vous déployez de nouveaux hôtes, le taux fixe est multiplié, ce qui complique le contrôle de la quantité de données enregistrées.

Activé AWS Lambda, vous ne pouvez pas modifier le taux d'échantillonnage. Si votre fonction est appelée par un service instrumenté, les appels ayant généré des demandes échantillonnées par ce service seront enregistrés par Lambda. Si le suivi actif est activé et qu'aucun en-tête de suivi n'est présent, Lambda prend la décision d'échantillonnage.

Pour configurer les règles de sauvegarde, demandez au SDK X-Ray pour Node.js de charger les règles d'échantillonnage à partir d'un fichier contenant `setSamplingRules`.

Exemple app.js - Règles d'échantillonnage à partir d'un fichier

```
var AWSXRay = require('aws-xray-sdk');
AWSXRay.middleware.setSamplingRules('sampling-rules.json');
```

Vous pouvez également définir des règles dans votre code et les transférer à `setSamplingRules` en tant qu'objet.

Exemple app.js - Échantillonnage des règles à partir d'un objet

```
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);
```

Pour utiliser uniquement des règles locales, appelez `disableCentralizedSampling`.

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

Journalisation

Pour vous connecter à partir du kit SDK, appelez `AWSXRay.setLogger(logger)`, où `logger` est un objet qui fournit des méthodes de journalisation standard (`warn`, `info`, etc.).

Par défaut, le SDK enregistre les messages d'erreur sur la console en utilisant les méthodes standard de l'objet de console. Le niveau de journalisation de l'enregistreur intégré peut être défini à l'aide des variables d'AWS_XRAY_LOG_LEVEL environnement AWS_XRAY_DEBUG_MODE ou des variables d'environnement. Pour obtenir la liste des valeurs de niveau de journalisation valides, consultez la section [Variables d'environnement](#).

Si vous souhaitez fournir un format ou une destination différents pour les journaux, vous pouvez fournir au SDK votre propre implémentation de l'interface de journalisation, comme indiqué ci-dessous. Tout objet implémentant cette interface peut être utilisé. Cela signifie que de nombreuses bibliothèques de journalisation, par exemple Winston, pourraient être utilisées et transmises directement au SDK.

Exemple app.js - journalisation

```
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]);
```

Appelez `setLogger` avant d'exécuter d'autres méthodes de configuration afin de veiller à capturer la sortie de ces opérations.

Adresse du daemon X-Ray

Si le daemon X-Ray écoute sur un port ou un hôte autre que celui-ci `127.0.0.1:2000`, vous pouvez configurer le SDK X-Ray pour Node.js afin d'envoyer les données de suivi à une adresse différente.

```
AWSXRay.setDaemonAddress('host:port');
```

Vous pouvez spécifier l'hôte par son nom ou son IPv4 adresse.

Exemple app.js - Adresse du démon

```
var AWSXRay = require('aws-xray-sdk');
AWSXRay.setDaemonAddress('daemonhost:8082');
```

Si vous avez configuré le démon de façon à ce qu'il écoute sur des ports différents pour les protocoles TCP et UDP, vous pouvez spécifier les deux dans l'adresse du démon.

Exemple app.js - Adresse du démon sur des ports distincts

```
var AWSXRay = require('aws-xray-sdk');
AWSXRay.setDaemonAddress('tcp:daemonhost:8082 udp:daemonhost:8083');
```

Vous pouvez également définir l'adresse du démon à l'aide de la `AWS_XRAY_DAEMON_ADDRESS` [variable d'environnement](#).

Variables d'environnement

Vous pouvez utiliser des variables d'environnement pour configurer le SDK X-Ray pour Node.js. Le kit SDK prend en charge les variables suivantes.

- `AWS_XRAY_CONTEXT_MISSING`— Réglé sur `RUNTIME_ERROR` pour générer des exceptions lorsque votre code instrumenté tente d'enregistrer des données alors qu'aucun segment n'est ouvert.

Valeurs valides

- `RUNTIME_ERROR`— Lance une exception d'exécution.
- `LOG_ERROR`— Enregistrez une erreur et continuez (par défaut).
- `IGNORE_ERROR`— Ignorez l'erreur et continuez.

Des erreurs liées à des segments ou sous-segments manquants peuvent se produire lorsque vous essayez d'utiliser un client instrumenté dans un code de démarrage qui s'exécute lorsqu'aucune demande n'est ouverte, ou dans un code qui génère un nouveau thread.

- `AWS_XRAY_DAEMON_ADDRESS`— Définissez l'hôte et le port de l'écouteur du daemon X-Ray. Par défaut, le SDK utilise à la fois `127.0.0.1:2000` les données de trace (UDP) et l'échantillonnage (TCP). Utilisez cette variable si vous avez configuré le démon pour qu'il [écoute sur un port différent](#) ou s'il s'exécute sur un autre hôte.

Format

- Même port — *address:port*
- Différents ports — `tcp:address:port` `udp:address:port`
- `AWS_XRAY_DEBUG_MODE`— Définissez sur `TRUE` pour configurer le SDK afin qu'il génère des journaux sur la console, au debug niveau 1.
- `AWS_XRAY_LOG_LEVEL` — Définissez un niveau de journalisation pour l'enregistreur par défaut. Les valeurs valides sont `debug`, `info`, `warn`, `error` et `silent`. Cette valeur est ignorée lorsqu'elle `AWS_XRAY_DEBUG_MODE` est définie sur `TRUE`.
- `AWS_XRAY_TRACING_NAME`— Définissez un nom de service que le SDK utilise pour les segments. Remplace le nom de segment que vous [définissez sur l'intergiciel Express](#).

Suivi des demandes entrantes avec le SDK X-Ray pour Node.js

Vous pouvez utiliser le SDK X-Ray pour Node.js afin de suivre les requêtes HTTP entrantes que vos applications Express et Restify traitent sur une EC2 instance Amazon EC2 ou Amazon ECS. AWS Elastic Beanstalk

Le SDK X-Ray pour Node.js fournit un intergiciel pour les applications qui utilisent les frameworks Express et Restify. Lorsque vous ajoutez le middleware X-Ray à votre application, le SDK X-Ray pour Node.js crée un segment pour chaque requête échantillonnée. Ce segment comprend la durée, la méthode et l'état de la demande HTTP. L'instrumentation supplémentaire crée des sous-segments sur ce segment.

Note

Pour les AWS Lambda fonctions, Lambda crée un segment pour chaque requête échantillonnée. Pour plus d'informations, consultez [AWS Lambda et AWS X-Ray](#).

Chaque segment porte un nom qui identifie votre application dans la carte des services. Le segment peut être nommé de manière statique ou vous pouvez configurer le SDK pour qu'il soit nommé dynamiquement en fonction de l'en-tête de l'hôte dans la demande entrante. La dénomination dynamique vous permet de regrouper les traces en fonction du nom de domaine indiqué dans la demande et d'appliquer un nom par défaut si le nom ne correspond pas au modèle attendu (par exemple, si l'en-tête de l'hôte est falsifié).

Demandes transmises

Si un équilibreur de charge ou un autre intermédiaire transmet une demande à votre application, X-Ray prend l'adresse IP du client depuis l'`X-Forwarded-For`-tête de la demande plutôt que depuis l'adresse IP source du paquet IP. L'adresse IP du client enregistrée pour une demande transférée peut être falsifiée, elle ne doit donc pas être fiable.

Lorsqu'une demande est transmise, le SDK définit un champ supplémentaire dans le segment pour l'indiquer. Si le segment contient le champ `x_forwarded_for` défini sur `true`, l'adresse IP du client a été extraite de l'`X-Forwarded-For`-tête de la requête HTTP.

Le gestionnaire de messages crée un segment pour chaque demande entrante avec un bloc `http` contenant les informations suivantes :

- Méthode HTTP : GET, POST, PUT, DELETE, etc.
- Adresse du client : adresse IP du client qui a envoyé la demande.
- Code de réponse : code de réponse HTTP pour la demande terminée.
- Moment : heure de début (date de réception de la demande) et heure de fin (date d'envoi de la réponse).
- Agent utilisateur — Le formulaire `user-agent` de la demande.
- Longueur du contenu : `content-length` extrait de la réponse.

Sections

- [Suivi des demandes entrantes avec Express](#)
- [Suivi des demandes entrantes avec Restify](#)
- [Configuration d'une stratégie d'attribution de noms de segment](#)

Suivi des demandes entrantes avec Express

Pour utiliser l'intégriciel Express, initialisez le client du kit de développement logiciel et utilisez l'intégriciel renvoyé par la fonction `express.openSegment` avant de définir vos routes.

Exemple 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());
```

Après avoir défini vos itinéraires, utilisez la sortie de `express.closeSegment` comme indiqué pour gérer les erreurs renvoyées par le SDK X-Ray pour Node.js.

Suivi des demandes entrantes avec Restify

Pour utiliser l'intergiciel Restify, initialisez le client du kit de développement logiciel et exécutez `enable`. Indiquez-lui votre serveur Restify et le nom de votre segment.

Exemple `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');
});
```

Configuration d'une stratégie d'attribution de noms de segment

AWS X-Ray utilise un nom de service pour identifier votre application et la distinguer des autres applications, bases de données, ressources externes APIs et AWS ressources utilisées par votre application. Lorsque le SDK X-Ray génère des segments pour les demandes entrantes, il enregistre le nom du service de votre application dans le [champ du nom](#) du segment.

Le SDK X-Ray peut nommer les segments d'après le nom d'hôte dans l'en-tête de la requête HTTP. Cependant, cet en-tête peut être falsifié, ce qui peut entraîner la création de nœuds inattendus dans votre carte de service. Pour éviter que le SDK ne nomme les segments de manière incorrecte en

raison de demandes contenant des en-têtes d'hôte falsifiés, vous devez spécifier un nom par défaut pour les demandes entrantes.

Si votre application traite des demandes pour plusieurs domaines, vous pouvez configurer le SDK pour qu'il utilise une stratégie de dénomination dynamique afin de refléter cela dans les noms des segments. Une stratégie de dénomination dynamique permet au SDK d'utiliser le nom d'hôte pour les demandes qui correspondent à un modèle attendu et d'appliquer le nom par défaut aux demandes qui ne le sont pas.

Par exemple, il se peut qu'une seule application envoie des demandes à trois sous-domaines : `www.example.com`, `api.example.com`, `etstatic.example.com`. Vous pouvez utiliser une stratégie de dénomination dynamique avec le modèle `*.example.com` pour identifier les segments de chaque sous-domaine avec un nom différent, ce qui permet d'obtenir trois nœuds de service sur la carte des services. Si votre application reçoit des demandes dont le nom d'hôte ne correspond pas au modèle, vous verrez un quatrième nœud sur la carte des services avec un nom de remplacement que vous spécifiez.

Pour utiliser le même nom pour tous les segments de la demande, spécifiez le nom de votre application lorsque vous initialisez l'intergiciel, comme indiqué dans les sections précédentes.

Note

Vous pouvez remplacer le nom de service par défaut que vous définissez avec la `AWS_XRAY_TRACING_NAME` variable d'environnement [???](#).

Une stratégie d'attribution de noms dynamique définit un modèle auquel doivent correspondre les noms d'hôte et un nom par défaut à utiliser si le nom d'hôte de la demande HTTP ne correspond pas au modèle. Pour nommer dynamiquement des segments, utilisez `AWSXRay.middleware.enableDynamicNaming`.

Exemple app.js - Noms de segment dynamiques

Si le nom d'hôte dans la demande correspond au modèle `*.example.com`, utilisez le nom d'hôte. Dans le cas contraire, utilisez `MyApp`.

```
var app = express();

var AWSXRay = require('aws-xray-sdk');
app.use(AWSXRay.express.openSegment('MyApp'));
```



```
AWSXRay.middleware.enableDynamicNaming('*.example.com');

app.get('/', function (req, res) {
  res.render('index');
});

app.use(AWSXRay.express.closeSegment());
```

Suivi des appels du AWS SDK avec le SDK X-Ray pour Node.js

Lorsque votre application effectue des appels Services AWS pour stocker des données, écrire dans une file d'attente ou envoyer des notifications, le SDK X-Ray pour Node.js suit les appels en aval dans des [sous-segments](#). Les ressources tracées Services AWS et auxquelles vous accédez au sein de ces services (par exemple, un compartiment Amazon S3 ou une file d'attente Amazon SQS) apparaissent sous forme de nœuds en aval sur la carte de trace de la console X-Ray.

Clients du AWS SDK d'instruments que vous créez via la [AWS SDK pour JavaScript V2](#) ou la [AWS SDK pour JavaScript V3](#). Chaque version du AWS SDK propose différentes méthodes pour instrumenter les clients du AWS SDK.

Note

Actuellement, le AWS X-Ray SDK pour Node.js renvoie moins d'informations sur les segments lors de l'instrumentation des clients AWS SDK pour JavaScript V3 que lors de l'instrumentation des clients V2. Par exemple, les sous-segments représentant des appels à DynamoDB ne renverront pas le nom de la table. Si vous avez besoin de ces informations de segment dans vos traces, pensez à utiliser le AWS SDK pour JavaScript V2.

AWS SDK pour JavaScript V2

Vous pouvez instrumenter tous les clients du AWS SDK V2 en encapsulant votre instruction `aws-sdk` require dans un appel à `AWSXRay.captureAWS`.

Exemple `app.js` - AWS Instrumentation du SDK

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

Pour instrumenter des clients individuels, associez votre client AWS SDK à un appel à `AWSXRay.captureAWSClient`. Par exemple, pour instrumenter un client `AmazonDynamoDB` :

Exemple app.js - Instrumentation du client DynamoDB

```
const AWSXRay = require('aws-xray-sdk');  
...  
const ddb = AWSXRay.captureAWSClient(new AWS.DynamoDB());
```

Warning

N'utilisez pas `captureAWS` et `captureAWSClient` ensemble. Ceci conduira à des sous-segments dupliqués.

Si vous souhaitez utiliser [TypeScript](#) des [ECMAScriptmodules](#) (ESM) pour charger votre JavaScript code, utilisez l'exemple suivant pour importer des bibliothèques :

Exemple app.js - AWS Instrumentation du SDK

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

Pour instrumenter tous les AWS clients avec ESM, utilisez le code suivant :

Exemple app.js - AWS Instrumentation du SDK

```
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();
```

Pour tous les services, vous pouvez voir le nom de l'API appelée dans la console X-Ray. Pour un sous-ensemble de services, le SDK X-Ray ajoute des informations au segment afin d'améliorer la granularité de la carte des services.

Par exemple, lorsque vous passez un appel avec un client DynamoDB instrumenté, le SDK ajoute le nom de la table au segment pour les appels qui ciblent une table. Dans la console, chaque table apparaît sous la forme d'un nœud distinct dans la carte des services, avec un nœud DynamoDB générique pour les appels qui ne ciblent pas une table.

Exemple Sous-segment pour un appel à DynamoDB pour enregistrer un élément

```
{
```

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

Lorsque vous accédez aux ressources nommées, les appels vers les services suivants créent des nœuds supplémentaires dans la cartographie des services. Les appels qui ne ciblent pas de ressources spécifiques créent un nœud générique pour le service.

- Amazon DynamoDB — Nom de la table
- Amazon Simple Storage Service : nom du compartiment et de la clé
- Amazon Simple Queue Service — Nom de la file d'attente

AWS SDK pour JavaScript V3

La AWS SDK pour JavaScript V3 étant modulaire, votre code ne charge que les modules dont il a besoin. De ce fait, il n'est pas possible d'instrumenter tous les clients du AWS SDK car la version 3 ne prend pas en charge cette méthode. `captureAWS`

Si vous souhaitez utiliser TypeScript des ECMAScript modules (ESM) pour charger votre JavaScript code, vous pouvez utiliser l'exemple suivant pour importer des bibliothèques :

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

Instrumentez chaque client du AWS SDK à l'aide de `AWSXRay.captureAWSSv3Client` cette méthode. Par exemple, pour instrumenter un client `AmazonDynamoDB` :

Exemple app.js - Instrumentation du client DynamoDB à l'aide du SDK pour Javascript V3

```
const AWSXRay = require('aws-xray-sdk');
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
...
const ddb = AWSXRay.captureAWSSv3Client(new DynamoDBClient({ region:
"region" })));
```

Lors de l'utilisation de la AWS SDK pour JavaScript V3, les métadonnées telles que le nom de la table, le nom du bucket et de la clé, ou le nom de la file d'attente, ne sont actuellement pas renvoyées. Par conséquent, la carte de trace ne contiendra pas de nœuds discrets pour chaque ressource nommée, comme elle le ferait lors de l'instrumentation des clients du AWS SDK à l'aide de la V2. AWS SDK pour JavaScript

Exemple Sous-segment pour un appel à DynamoDB pour enregistrer un élément, lors de l'utilisation de la V3 AWS SDK pour 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",
  }
}
```

Suivi des appels vers les services Web HTTP en aval à l'aide du SDK X-Ray pour Node.js

Lorsque votre application appelle des microservices ou du protocole HTTP public APIs, vous pouvez utiliser le client X-Ray SDK for Node.js pour instrumenter ces appels et ajouter l'API au graphe de service en tant que service en aval.

Transmettez votre `http` ou votre `https` client au SDK X-Ray pour la `captureHTTPs` méthode Node.js afin de suivre les appels sortants.

Note

Les appels utilisant des bibliothèques de demandes HTTP tierces, telles qu'Axios ou Superagent, sont pris en charge via l'[API `captureHTTPsGlobal\(\)`](#) et seront toujours suivis lorsqu'ils utilisent le module `http` natif.

Exemple app.js - Client HTTP

```
var AWSXRay = require('aws-xray-sdk');
var http = AWSXRay.captureHTTPs(require('http'));
```

Pour activer le suivi sur tous les clients HTTP, appelez `captureHTTPsGlobal` avant de charger `http`.

Exemple app.js - Client HTTP (global)

```
var AWSXRay = require('aws-xray-sdk');
AWSXRay.captureHTTPsGlobal(require('http'));
var http = require('http');
```

Lorsque vous instrumentez un appel à une API Web en aval, le SDK X-Ray pour Node.js enregistre un sous-segment contenant des informations sur la requête et la réponse HTTP. X-Ray utilise le sous-segment pour générer un segment inféré pour l'API distante.

Exemple Sous-segment pour un appel HTTP en aval

```
{
  "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
  }
}
```

Exemple Segment déduit pour un appel HTTP en aval

```
{
  "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
}
```

Suivi des requêtes SQL avec le SDK X-Ray pour Node.js

Instrumentez les requêtes de base de données SQL en encapsulant votre client SQL dans la méthode cliente X-Ray SDK for Node.js correspondante.

- 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);
```

Lorsque vous utilisez un client instrumenté pour effectuer des requêtes SQL, le kit SDK X-Ray pour Node.js enregistre les informations sur la connexion et la requête dans un sous-segment.

Inclusion de données supplémentaires dans les sous-segments SQL

Vous pouvez ajouter des informations supplémentaires aux sous-segments générés pour les requêtes SQL, à condition qu'elles soient mappées à un champ SQL autorisé. Par exemple, pour enregistrer la chaîne de requête SQL nettoyée dans un sous-segment, vous pouvez l'ajouter directement à l'objet SQL du sous-segment.

Exemple Affecter du code SQL à un sous-segment

```
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;
}
```

Pour obtenir la liste complète des champs SQL autorisés, consultez la section [Requêtes SQL](#) du manuel du AWS X-Ray développeur.

Génération de sous-segments personnalisés avec le SDK X-Ray pour Node.js

Les sous-segments étendent le [segment](#) d'une trace avec des détails sur le travail effectué afin de répondre à une demande. Chaque fois que vous passez un appel avec un client instrumenté, le SDK X-Ray enregistre les informations générées dans un sous-segment. Vous pouvez créer des sous-segments supplémentaires pour regrouper d'autres sous-segments, pour mesurer les performances d'une section de code ou pour enregistrer des annotations et des métadonnées.

Sous-segments Express personnalisés

Pour créer un sous-segment personnalisé pour une fonction qui effectue des appels vers les services, utilisez la fonction `captureAsyncFunc`.

Exemple app.js - Sous segments personnalisés 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();
};
```

Dans cet exemple, l'application crée un sous-segment personnalisé nommé `send` pour les appels vers la fonction `sendRequest`. `captureAsyncFunc` transmet un sous-segment que vous devez fermer dans la fonction de rappel lorsque les appels asynchrones qu'elle effectue sont terminés.

Pour les fonctions synchrones, vous pouvez utiliser la fonction `captureFunc` qui ferme automatiquement le sous-segment dès la fin de l'exécution du bloc de fonction.

Lorsque vous créez un sous-segment au sein d'un segment ou d'un autre sous-segment, le SDK X-Ray pour Node.js génère un identifiant et enregistre l'heure de début et de fin.

Exemple Sous-segment avec des métadonnées

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

Sous-segments Lambda personnalisés

Le SDK est configuré pour créer automatiquement un segment de façade fictif lorsqu'il détecte qu'il s'exécute dans Lambda. Pour créer un sous-segment de base, qui créera un

AWS::Lambda::Function nœud unique sur la carte de trace X-Ray, appelez et réutilisez le segment de façade. Si vous créez manuellement un nouveau segment avec un nouvel ID (tout en partageant l'ID de suivi, l'ID parent et la décision d'échantillonnage), vous pourrez envoyer un nouveau segment.

Exemple app.js - sous-segments personnalisés manuels

```
const segment = AWSXRay.getSegment(); //returns the facade segment
const subsegment = segment.addNewSubsegment('subseg');
...
subsegment.close();
//the segment is closed by the SDK automatically
```

Ajoutez des annotations et des métadonnées aux segments avec le SDK X-Ray pour Node.js

Vous pouvez enregistrer des informations supplémentaires sur les demandes, l'environnement ou votre application à l'aide d'annotations et de métadonnées. Vous pouvez ajouter des annotations et des métadonnées aux segments créés par le SDK X-Ray ou aux sous-segments personnalisés que vous créez.

Les annotations sont des paires clé-valeur avec des chaînes, des nombres ou des valeurs booléennes. Les annotations sont indexées pour être utilisées avec les expressions de [filtre](#). Utilisez les annotations pour enregistrer les données que vous souhaitez utiliser pour regrouper les suivis dans la console ou lors de l'appel de l'API [GetTraceSummaries](#).

Les métadonnées sont des paires clé-valeur qui peuvent contenir des valeurs de n'importe quel type, y compris des objets et des listes, mais qui ne sont pas indexées pour être utilisées avec des expressions de filtre. Utilisez les métadonnées pour enregistrer des données supplémentaires que vous souhaitez stocker dans le traçage, mais dont vous n'avez pas besoin pour effectuer une recherche.

En plus des annotations et des métadonnées, vous pouvez également [enregistrer les chaînes d'ID utilisateur](#) sur des segments. Les utilisateurs sont enregistrés dans un champ distinct sur les segments et sont indexés pour être utilisés dans le cadre de la recherche.

Sections

- [Enregistrement d'annotations avec le SDK X-Ray pour Node.js](#)

- [Enregistrement de métadonnées avec le SDK X-Ray pour Node.js](#)
- [Utilisateur enregistrant IDs avec le SDK X-Ray pour Node.js](#)

Enregistrement d'annotations avec le SDK X-Ray pour Node.js

Utilisez les annotations pour enregistrer les informations sur les segments ou sous-segments qui doivent être indexés pour la recherche.

Exigences liées aux annotations

- Clés — La clé d'une annotation X-Ray peut comporter jusqu'à 500 caractères alphanumériques. Vous ne pouvez pas utiliser d'espaces ou de symboles autres qu'un point ou un point (.)
- Valeurs — La valeur d'une annotation X-Ray peut comporter jusqu'à 1 000 caractères Unicode.
- Nombre d'annotations : vous pouvez utiliser jusqu'à 50 annotations par trace.

Pour enregistrer les annotations

1. Obtenez une référence au segment ou sous-segment en cours.

```
var AWSXRay = require('aws-xray-sdk');  
...  
var document = AWSXRay.getSegment();
```

2. Appelez `addAnnotation` avec une clé de type chaîne et une valeur de type booléen, numérique ou chaîne.

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

L'exemple suivant montre comment appeler `putAnnotation` avec une clé String qui inclut un point et une valeur booléenne, numérique ou de chaîne.

```
document.putAnnotation("testkey.test", "my value");
```

Le kit de développement logiciel enregistre les annotations sous forme de paires clé-valeur dans un objet `annotations` du document de segment. Si vous appelez deux fois `addAnnotation` avec la même clé, les valeurs précédemment enregistrées sur le même segment ou sous-segment sont remplacées.

Pour rechercher les suivis ayant des annotations avec des valeurs spécifiques, utilisez le mot clé `annotation[key]` dans une [expression de filtre](#).

Exemple app.js - Annotations

```
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) {
    ...
  });
});
```

Enregistrement de métadonnées avec le SDK X-Ray pour Node.js

Utilisez les métadonnées pour enregistrer des informations sur les segments ou sous-segments qui n'ont pas besoin d'être indexés pour la recherche. Les valeurs des métadonnées peuvent être de type chaîne, valeur numérique, valeur booléenne, ou tout autre objet à même d'être sérialisé en un tableau ou objet JSON.

Pour enregistrer les métadonnées

1. Obtenez une référence au segment ou sous-segment en cours.

```
var AWSXRay = require('aws-xray-sdk');
...
var document = AWSXRay.getSegment();
```

2. Appelez `addMetadata` avec une clé de type chaîne, une valeur de type booléen, numérique, chaîne ou objet et un espace de noms de type chaîne.

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

or

Appelez `addMetadata` avec seulement une clé et une valeur.

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

Si vous ne spécifiez pas d'espace de noms, le kit de développement logiciel utilise `default`. Si vous appelez deux fois `addMetadata` avec la même clé, les valeurs précédemment enregistrées sur le même segment ou sous-segment sont remplacées.

Utilisateur enregistrant IDs avec le SDK X-Ray pour Node.js

Enregistrez les segments utilisateur IDs sur demande pour identifier l'utilisateur qui a envoyé la demande. Cette opération n'est pas compatible avec AWS Lambda les fonctions car les segments dans les environnements Lambda sont immuables. L'appel `setUser` peut être appliqué uniquement aux segments, pas aux sous-segments.

Pour enregistrer un utilisateur IDs

1. Obtenez une référence au segment ou sous-segment en cours.

```
var AWSXRay = require('aws-xray-sdk');  
...  
var document = AWSXRay.getSegment();
```

2. Appelez `setUser()` avec l'ID de type chaîne de l'utilisateur ayant envoyé la demande.

```
var user = 'john123';  
  
AWSXRay.getSegment().setUser(user);
```

Vous pouvez appeler `setUser` pour enregistrer l'ID utilisateur dès que votre application express commence à traiter une demande. Si vous n'utilisez le segment que pour définir l'ID utilisateur, vous pouvez chaîner les appels sur une seule ligne.

Exemple app.js - ID utilisateur

```
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) {
    ...
  });
});
```

Pour rechercher les suivis d'un ID utilisateur, utilisez le mot clé `user` dans une [expression de filtre](#).

Travail avec Python

Il existe deux manières d'instrumenter votre application Python pour envoyer des traces à X-Ray :

- [AWS Distro pour OpenTelemetry Python](#) : AWS distribution qui fournit un ensemble de bibliothèques open source permettant d'envoyer des métriques et des traces corrélées à plusieurs solutions de AWS surveillance, notamment Amazon et Amazon OpenSearch Service CloudWatch AWS X-Ray, via [AWS Distro](#) for Collector. OpenTelemetry
- [AWS X-Ray SDK pour Python](#) : ensemble de bibliothèques permettant de générer et d'envoyer des traces à X-Ray via le [daemon X-Ray](#).

Pour de plus amples informations, veuillez consulter [Choisir entre The AWS Distro for OpenTelemetry et X-Ray SDKs](#).

AWS Distro pour Python OpenTelemetry

Avec le Python AWS Distro for OpenTelemetry (ADOT), vous pouvez instrumenter vos applications une seule fois et envoyer des métriques et des traces corrélées à plusieurs solutions de AWS surveillance CloudWatch, notamment Amazon AWS X-Ray et Amazon Service. OpenSearch L'utilisation de X-Ray avec ADOT nécessite deux composants : un OpenTelemetry SDK activé pour une utilisation avec X-Ray, et le AWS Distro for OpenTelemetry Collector activé pour une utilisation avec X-Ray. ADOT Python inclut la prise en charge de l'instrumentation automatique, permettant à votre application d'envoyer des traces sans modifier le code.

Pour commencer, consultez la [documentation de AWS Distro for OpenTelemetry Python](#).

Pour plus d'informations sur l'utilisation du AWS Distro pour OpenTelemetry with AWS X-Ray et d'autres Services AWS, voir [AWS Distro for OpenTelemetry](#) ou [AWS Distro pour la documentation](#). OpenTelemetry

Pour plus d'informations sur la prise en charge et l'utilisation des langues, consultez [AWS Observability on GitHub](#).

AWS X-Ray SDK pour Python

Le SDK X-Ray pour Python est une bibliothèque pour les applications Web Python qui fournit des classes et des méthodes permettant de générer et d'envoyer des données de trace au démon

X-Ray. Les données de suivi incluent des informations sur les requêtes HTTP entrantes traitées par l'application et les appels que l'application envoie aux services en aval à l'aide du AWS SDK, des clients HTTP ou d'un connecteur de base de données SQL. Vous pouvez également créer des segments manuellement et ajouter des informations de débogage dans les annotations et les métadonnées.

Vous pouvez télécharger le kit SDK avec pip.

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

Note

Le SDK X-Ray pour Python est un projet open source. Vous pouvez suivre le projet et soumettre des problèmes et des pull requests sur GitHub : github.com/aws/aws-xray-sdk-python

Si vous utilisez Django ou Flask, commencez par [ajouter l'intergiciel du SDK à votre application](#) afin de suivre les demandes entrantes. L'intergiciel crée un [segment](#) pour chaque demande suivie et termine le segment lorsque la réponse est envoyée. Lorsque le segment est ouvert, vous pouvez utiliser les méthodes du client du SDK pour ajouter des informations au segment et créer des sous-segments afin de suivre les appels en aval. Le kit de développement logiciel (SDK) enregistre aussi automatiquement les exceptions que votre application renvoie quand le segment est ouvert. Pour les autres applications, vous pouvez [créer des segments manuellement](#).

Pour les fonctions Lambda appelées par une application ou un service instrumenté, Lambda lit l'[entête de suivi et trace automatiquement les requêtes échantillonnées](#). Pour les autres fonctions, vous pouvez [configurer Lambda](#) pour échantillonner et suivre les demandes entrantes. Dans les deux cas, Lambda crée le segment et le fournit au SDK X-Ray.

Note

Sur Lambda, le SDK X-Ray est facultatif. Si vous ne l'utilisez pas dans votre fonction, votre carte de service inclura toujours un nœud pour le service Lambda et un nœud pour chaque fonction Lambda. En ajoutant le SDK, vous pouvez instrumenter votre code de fonction pour ajouter des sous-segments au segment de fonction enregistré par Lambda. Pour plus d'informations, consultez [AWS Lambda et AWS X-Ray](#).

Voir un [Nœuds](#) exemple de fonction Python instrumentée dans Lambda.

Utilisez ensuite le SDK X-Ray pour Python pour instrumenter les appels en aval en appliquant des [correctifs aux bibliothèques de votre application](#). Le kit SDK prend en charge les bibliothèques suivantes :

Bibliothèques prises en charge

- [botocore](#), [boto3](#) — AWS SDK for Python (Boto) Clients de l'instrument.
- [pynamodb](#)— Version du client Amazon DynamoDB de l'instrument PynamoDB.
- [aiobotocore](#), [aioboto3](#) — Versions intégrées à Instrument [asyncio](#) du SDK pour les clients Python.
- [requests](#), [aiohttp](#) — Clients HTTP de haut niveau de l'instrument.
- [httplib](#), [http.client](#)— Instrument les clients HTTP de bas niveau et les bibliothèques de niveau supérieur qui les utilisent.
- [sqlite3](#)— SQLite Clients d'instruments.
- [mysql-connector-python](#)— Clients MySQL d'instruments.
- [pg8000](#)— Interface PostgreSQL en Python pur de l'instrument.
- [psycopg2](#)— Adaptateur de base de données PostgreSQL Instrument.
- [pymongo](#)— Instrument les clients MongoDB.
- [pymysql](#)— Clients basés sur Instrument PyMy SQL pour MySQL et MariaDB.

Chaque fois que votre application appelle une base de données SQL ou d'autres services HTTP, le SDK enregistre les informations relatives à l'appel dans un sous-segment. AWS Services AWS et les ressources auxquelles vous accédez au sein des services apparaissent sous forme de nœuds en aval sur la carte de trace pour vous aider à identifier les erreurs et les problèmes de limitation sur les connexions individuelles.

Après avoir commencé à utiliser le SDK, personnalisez son comportement en [configurant l'enregistreur et le middleware](#). Vous pouvez ajouter des plug-ins pour enregistrer les données sur les ressources de calcul exécutant votre application, personnaliser le comportement d'échantillonnage en définissant des règles d'échantillonnage et définir le niveau de journal afin d'afficher plus ou moins d'informations du kit de développement logiciel dans vos journaux d'application.

Enregistrez les informations supplémentaires sur les demandes et le travail que votre application effectue dans les [annotations et les métadonnées](#). Les annotations sont de simples paires clé-valeur,

indexées en vue de leur utilisation avec les [expressions de filtre](#), de telle sorte que vous pouvez explorer les suivis qui contiennent des données spécifiques. Les entrées des métadonnées sont moins restrictives et peuvent enregistrer des objets et tableaux entiers (tout ce qui peut être sérialisé en JSON).

Annotations et métadonnées

Les annotations et les métadonnées sont du texte arbitraire que vous ajoutez aux segments avec le SDK X-Ray. Les annotations sont indexées pour être utilisées avec les expressions de filtre. Les métadonnées ne sont pas indexées, mais peuvent être consultées dans le segment brut à l'aide de la console ou de l'API X-Ray. Toute personne à qui vous accordez un accès en lecture à X-Ray peut consulter ces données.

Lorsque vous avez un grand nombre de clients instrumentés dans votre code, un seul segment de demande peut contenir un grand nombre de sous-segments, un par appel effectué à l'aide d'un client instrumenté. Vous pouvez organiser et grouper les sous-segments en enveloppant les appels clients dans des [sous-segments personnalisés](#). Vous pouvez créer un sous-segment personnalisé pour la totalité d'une fonction ou pour une section de code. Vous pouvez ensuite enregistrer des métadonnées et des annotations sur le sous-segment au lieu de tout écrire sur le segment parent.

Pour consulter la documentation de référence sur les classes et les méthodes du SDK, consultez le manuel de référence des [API du AWS X-Ray SDK pour Python](#).

Prérequis

Le SDK X-Ray pour Python prend en charge les versions de langage et de bibliothèque suivantes.

- Python — 2.7, 3.4 et versions ultérieures
- Django — 1.10 et versions ultérieures
- Flask — 0.10 et versions ultérieures
- aiohttp — 2.3.0 et versions ultérieures
- AWS SDK for Python (Boto)— 1.4.0 et versions ultérieures
- botocore — 1.5.0 et versions ultérieures
- enum — 0.4.7 et versions ultérieures, pour les versions 3.4.0 et antérieures de Python
- jsonpickle — 1.0.0 et versions ultérieures
- setuptools — 40.6.3 et versions ultérieures

- `wrapt` — 1.11.0 et versions ultérieures

Gestion des dépendances

Le SDK X-Ray pour Python est disponible sur `pip`

- Package — `aws-xray-sdk`

Ajoutez le kit SDK en tant que dépendance dans le fichier `requirements.txt`.

Exemple `requirements.txt`

```
aws-xray-sdk==2.4.2
boto3==1.4.4
botocore==1.5.55
Django==1.11.3
```

Si vous utilisez Elastic Beanstalk pour déployer votre application, Elastic Beanstalk installe automatiquement tous les packages. `requirements.txt`

Configuration du kit de développement X-Ray pour Python

Le SDK X-Ray pour Python possède une classe nommée `xray_recorder` qui fournit l'enregistreur global. Vous pouvez configurer l'enregistreur mondial afin qu'il personnalise l'intergiciel qui crée des segments pour les appels HTTP entrants.

Sections

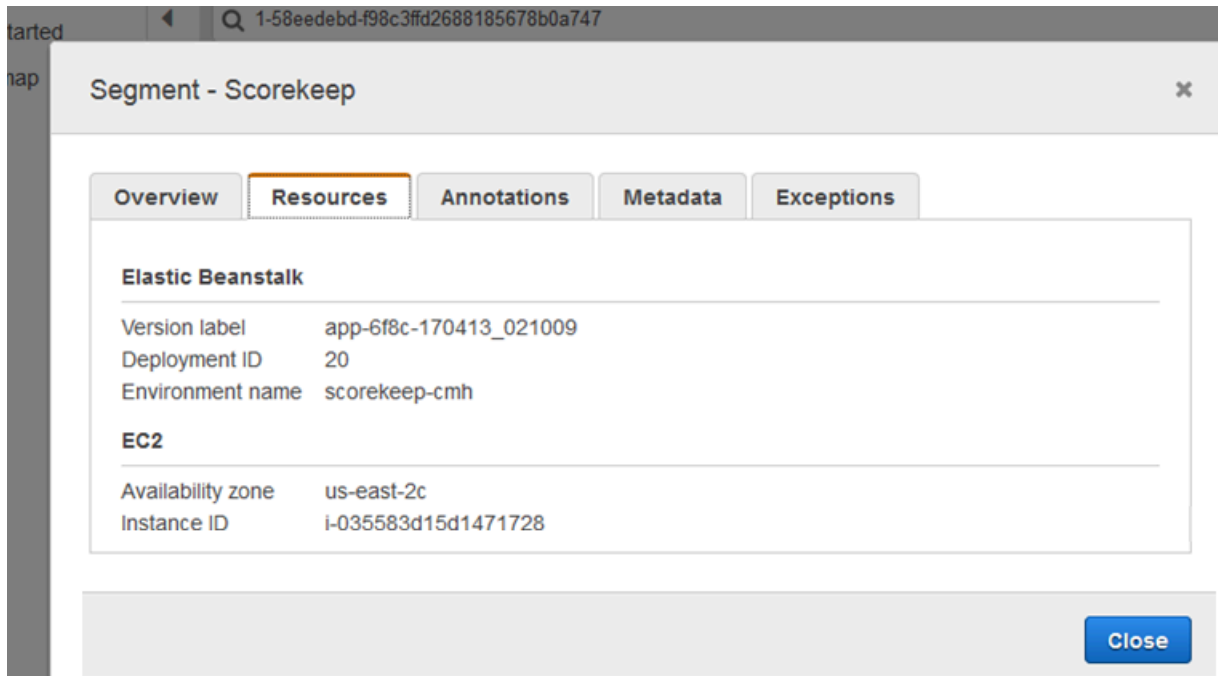
- [Plug-ins de service](#)
- [Règles d'échantillonnage](#)
- [Journalisation](#)
- [Configuration de l'enregistreur dans le code](#)
- [Configuration de l'enregistreur avec Django](#)
- [Variables d'environnement](#)

Plug-ins de service

Permet plugins d'enregistrer des informations sur le service hébergeant votre application.

Plugins

- Amazon EC2 — EC2Plugin ajoute l'ID de l'instance, la zone de disponibilité et le groupe de CloudWatch journaux.
- Elastic BeanstalkPlugin Beanstalk : ajoute le nom de l'environnement, l'étiquette de version et l'ID de déploiement.
- Amazon ECS — ECSPlugin ajoute l'ID du conteneur.



Pour utiliser un plug-in, appelez `configure` sur l'enregistreur `xray_recorder`.

```
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

Comme `plugins` ils sont transmis sous forme de tuple, veuillez à inclure une fin `,` lorsque vous spécifiez un seul plugin. Par exemple, `plugins = ('EC2Plugin',)`

Vous pouvez également utiliser des [variables d'environnement](#), qui ont la priorité sur les valeurs définies dans le code, pour configurer l'enregistreur.

Configurez les plug-ins avant les [bibliothèques de correctifs](#) pour enregistrer les appels en aval.

Le SDK utilise également les paramètres du plugin pour définir le `origin` champ du segment. Cela indique le type de AWS ressource qui exécute votre application. Lorsque vous utilisez plusieurs plug-ins, le SDK utilise l'ordre de résolution suivant pour déterminer l'origine : ElasticBeanstalk > EKS > ECS > EC2.

Règles d'échantillonnage

Le SDK utilise les règles d'échantillonnage que vous définissez dans la console X-Ray pour déterminer les demandes à enregistrer. La règle par défaut suit la première demande chaque seconde, et 5 % de toutes les demandes supplémentaires provenant de tous les services envoient des traces à X-Ray. [Créez des règles supplémentaires dans la console X-Ray](#) pour personnaliser la quantité de données enregistrées pour chacune de vos applications.

Le SDK applique les règles personnalisées dans l'ordre dans lequel elles sont définies. Si une demande correspond à plusieurs règles personnalisées, le SDK applique uniquement la première règle.

Note

Si le SDK ne parvient pas à accéder à X-Ray pour obtenir des règles d'échantillonnage, il revient à une règle locale par défaut concernant la première demande chaque seconde, et 5 % de toutes les demandes supplémentaires par hôte. Cela peut se produire si l'hôte n'est pas autorisé à appeler `sampling APIs` ou ne peut pas se connecter au daemon X-Ray, qui agit comme un proxy TCP pour les appels d'API effectués par le SDK.

Vous pouvez également configurer le SDK pour charger des règles d'échantillonnage à partir d'un document JSON. Le SDK peut utiliser les règles locales comme solution de rechange dans les cas où l'échantillonnage X-Ray n'est pas disponible, ou utiliser exclusivement les règles locales.

Exemple `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
}
```

Cet exemple définit une règle personnalisée et une règle par défaut. La règle personnalisée applique un taux d'échantillonnage de 5 % sans nombre minimum de demandes à suivre pour les chemins sous-jacents. `/api/move/` La règle par défaut suit la première demande chaque seconde et 10 % des demandes supplémentaires.

L'inconvénient de définir des règles localement est que la cible fixe est appliquée par chaque instance de l'enregistreur indépendamment, au lieu d'être gérée par le service X-Ray. Au fur et à mesure que vous déployez de nouveaux hôtes, le taux fixe est multiplié, ce qui complique le contrôle de la quantité de données enregistrées.

Activé AWS Lambda, vous ne pouvez pas modifier le taux d'échantillonnage. Si votre fonction est appelée par un service instrumenté, les appels ayant généré des demandes échantillonnées par ce service seront enregistrés par Lambda. Si le suivi actif est activé et qu'aucun en-tête de suivi n'est présent, Lambda prend la décision d'échantillonnage.

Pour configurer les règles d'échantillonnage de `sauvegardexray_recorder.configure`, appelez, comme indiqué dans l'exemple suivant, où se *rules* trouve soit un dictionnaire de règles, soit le chemin absolu vers un fichier JSON contenant des règles d'échantillonnage.

```
xray_recorder.configure(sampling_rules=rules)
```

Pour utiliser uniquement les règles locales, configurez l'enregistreur avec une instruction `LocalSampler`.

```
from aws_xray_sdk.core.sampling.local.sampler import LocalSampler
```

```
xray_recorder.configure(sampler=LocalSampler())
```

Vous pouvez également configurer l'enregistreur mondial afin qu'il désactive l'échantillonnage et instrumente toutes les demandes entrantes.

Exemple main.py — Désactive l'échantillonnage

```
xray_recorder.configure(sampling=False)
```

Journalisation

Le SDK utilise le logging module intégré de Python avec un niveau de WARNING journalisation par défaut. Obtenez une référence à l'enregistreur d'événements pour la classe `aws_xray_sdk` et appelez `setLevel` sur ce dernier pour configurer le niveau de journalisation différent pour la bibliothèque et le reste de l'application.

Exemple app.py — Journalisation

```
logging.basicConfig(level='WARNING')
logging.getLogger('aws_xray_sdk').setLevel(logging.ERROR)
```

Utilisez les journaux de débogage pour identifier les problèmes, tels que des sous-segments ouverts, lorsque vous [générez manuellement des sous-segments](#).

Configuration de l'enregistreur dans le code

D'autres paramètres sont disponibles à partir de la méthode `configure` dans `xray_recorder`.

- `context_missing`— Réglé sur `LOG_ERROR` pour éviter de générer des exceptions lorsque votre code instrumenté tente d'enregistrer des données alors qu'aucun segment n'est ouvert.
- `daemon_address`— Définissez l'hôte et le port de l'écouteur du daemon X-Ray.
- `service`— Définissez un nom de service que le SDK utilise pour les segments.
- `plugins`— Enregistrez des informations sur les AWS ressources de votre application.
- `sampling`— Réglez sur `False` pour désactiver l'échantillonnage.
- `sampling_rules`— Définissez le chemin du fichier JSON contenant vos [règles d'échantillonnage](#).

Exemple main.py — Désactive les exceptions manquantes dans le contexte

```
from aws_xray_sdk.core import xray_recorder
```

```
xray_recorder.configure(context_missing='LOG_ERROR')
```

Configuration de l'enregistreur avec Django

Si vous utilisez l'infrastructure Django, vous pouvez utiliser le fichier Django `settings.py` pour configurer les options de l'enregistreur mondial.

- `AUTO_INSTRUMENT`(Django uniquement) — Enregistrez des sous-segments pour les opérations de rendu de base de données et de modèles intégrées.
- `AWS_XRAY_CONTEXT_MISSING`— Réglé sur `LOG_ERROR` pour éviter de générer des exceptions lorsque votre code instrumenté tente d'enregistrer des données alors qu'aucun segment n'est ouvert.
- `AWS_XRAY_DAEMON_ADDRESS`— Définissez l'hôte et le port de l'écouteur du daemon X-Ray.
- `AWS_XRAY_TRACING_NAME`— Définissez un nom de service que le SDK utilise pour les segments.
- `PLUGINS`— Enregistrez des informations sur les AWS ressources de votre application.
- `SAMPLING`— Réglez sur `False` pour désactiver l'échantillonnage.
- `SAMPLING_RULES`— Définissez le chemin du fichier JSON contenant vos [règles d'échantillonnage](#).

Pour activer la configuration de l'enregistreur dans `settings.py`, ajoutez l'intergiciel Django à la liste des applications installées.

Exemple settings.py — Applications installées

```
INSTALLED_APPS = [  
    ...  
    'django.contrib.sessions',  
    'aws_xray_sdk.ext.django',  
]
```

Configurez les paramètres disponibles dans un dictionnaire nommé `XRAY_RECORDER`.

Exemple settings.py — Applications installées

```
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,
}
```

Variables d'environnement

Vous pouvez utiliser des variables d'environnement pour configurer le SDK X-Ray pour Python. Le kit SDK prend en charge les variables suivantes:

- **AWS_XRAY_TRACING_NAME**— Définissez un nom de service que le SDK utilise pour les segments. Remplace le nom du service que vous avez défini par programmation.
- **AWS_XRAY_SDK_ENABLED**— Lorsqu'il est défini sur `false`, le SDK est désactivé. Par défaut, le kit SDK est activé, sauf si la variable d'environnement est définie sur `false`.
 - Lorsque cette option est désactivée, l'enregistreur mondial génère automatiquement des segments et des sous-segments fictifs qui ne sont pas envoyés au démon, et l'application automatique des correctifs est désactivée. Les intergiciels sont écrits en tant que wrapper sur l'enregistreur mondial. Tous les segments et sous-segments générés via l'intergiciel deviennent également des segments et sous-segments factices .
 - Définissez la valeur de **AWS_XRAY_SDK_ENABLED** via la variable d'environnement ou via une interaction directe avec l'objet `global_sdk_config` de la bibliothèque `aws_xray_sdk`. Les paramètres définis dans la variable d'environnement remplacent ces interactions.
- **AWS_XRAY_DAEMON_ADDRESS**— Définissez l'hôte et le port de l'écouteur du daemon X-Ray. Par défaut, le SDK utilise à la fois `127.0.0.1:2000` les données de trace (UDP) et l'échantillonnage (TCP). Utilisez cette variable si vous avez configuré le démon pour qu'il [écoute sur un port différent](#) ou s'il s'exécute sur un autre hôte.

Format

- Même port — *address:port*
- Différents ports — `tcp:address:port` `udp:address:port`
- **AWS_XRAY_CONTEXT_MISSING**— Réglé sur `RUNTIME_ERROR` pour générer des exceptions lorsque votre code instrumenté tente d'enregistrer des données alors qu'aucun segment n'est ouvert.

Valeurs valides

- **RUNTIME_ERROR**— Lance une exception d'exécution.

- LOG_ERROR— Enregistrez une erreur et continuez (par défaut).
- IGNORE_ERROR— Ignorez l'erreur et continuez.

Des erreurs liées à des segments ou sous-segments manquants peuvent se produire lorsque vous essayez d'utiliser un client instrumenté dans un code de démarrage qui s'exécute lorsqu'aucune demande n'est ouverte, ou dans un code qui génère un nouveau thread.

Les variables d'environnement remplacent les valeurs définies dans le code.

Suivi des demandes entrantes avec le kit de développement X-Ray pour le middleware Python

Lorsque vous ajoutez le middleware à votre application et que vous configurez un nom de segment, le SDK X-Ray pour Python crée un segment pour chaque requête échantillonnée. Ce segment comprend la durée, la méthode et l'état de la demande HTTP. L'instrumentation supplémentaire crée des sous-segments sur ce segment.

Le SDK X-Ray pour Python prend en charge les intergiciels suivants pour instrumenter les requêtes HTTP entrantes :

- Django
- Flask
- Bottle

Note

Pour les AWS Lambda fonctions, Lambda crée un segment pour chaque requête échantillonnée. Pour plus d'informations, consultez [AWS Lambda et AWS X-Ray](#).

Voir un [Noeuds](#) exemple de fonction Python instrumentée dans Lambda.

Pour les scripts ou applications Python sur d'autres frameworks, vous pouvez [créer des segments manuellement](#).

Chaque segment porte un nom qui identifie votre application dans la carte des services. Le segment peut être nommé de manière statique ou vous pouvez configurer le SDK pour qu'il soit nommé

dynamiquement en fonction de l'en-tête de l'hôte dans la demande entrante. La dénomination dynamique vous permet de regrouper les traces en fonction du nom de domaine indiqué dans la demande et d'appliquer un nom par défaut si le nom ne correspond pas au modèle attendu (par exemple, si l'en-tête de l'hôte est falsifié).

Demandes transmises

Si un équilibreur de charge ou un autre intermédiaire transmet une demande à votre application, X-Ray prend l'adresse IP du client depuis l'`X-Forwarded-For`-tête de la demande plutôt que depuis l'adresse IP source du paquet IP. L'adresse IP du client enregistrée pour une demande transférée peut être falsifiée, elle ne doit donc pas être fiable.

Lorsqu'une demande est transmise, le SDK définit un champ supplémentaire dans le segment pour l'indiquer. Si le segment contient le champ `x_forwarded_for` défini sur `true`, l'adresse IP du client a été extraite de l'`X-Forwarded-For`-tête de la requête HTTP.

L'intergiciel crée un segment pour chaque demande entrante avec un bloc `http` contenant les informations suivantes :

- Méthode HTTP : GET, POST, PUT, DELETE, etc.
- Adresse du client : adresse IP du client qui a envoyé la demande.
- Code de réponse : code de réponse HTTP pour la demande terminée.
- Moment : heure de début (date de réception de la demande) et heure de fin (date d'envoi de la réponse).
- Agent utilisateur — Le formulaire `user-agent` de la demande.
- Longueur du contenu : `content-length` extrait de la réponse.

Sections

- [Ajout des intergiciels à votre application \(Django\)](#)
- [Ajout des intergiciels à votre application \(Flask\)](#)
- [Ajout des intergiciels à votre application \(Bottle\)](#)
- [Instrumentation manuelle du code Python](#)
- [Configuration d'une stratégie d'attribution de noms de segment](#)

Ajout des intergiciels à votre application (Django)

Ajoutez les intergiciels à la liste MIDDLEWARE dans le fichier `settings.py`. L'intergiciel X-Ray doit figurer sur la première ligne du fichier `settings.py` afin que les demandes qui échouent dans d'autres intergiciels soient enregistrées.

Exemple `settings.py` - SDK X-Ray pour intergiciel Python

```
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'  
]
```

Ajoutez l'application Django du SDK X-Ray à la `INSTALLED_APPS` liste de votre `settings.py` fichier. Cela permettra de configurer l'enregistreur X-Ray lors du démarrage de votre application.

Exemple `settings.py` - SDK X-Ray pour l'application Python Django

```
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',  
]
```

Configurez un nom de segment dans le fichier [settings.py](#).

Exemple `settings.py` — Nom du segment

```
XRAY_RECORDER = {  
    'AWS_XRAY_TRACING_NAME': 'My application',  
    'PLUGINS': ('EC2Plugin',),  
}
```

Cela indique à l'enregistreur X-Ray de suivre les requêtes traitées par votre application Django avec le taux d'échantillonnage par défaut. Vous pouvez [configurer les paramètres du fichier Django de l'enregistreur](#) pour appliquer les règles d'échantillonnage personnalisées ou modifier d'autres paramètres.

Note

Comme plugins ils sont transmis sous forme de tuple, veuillez à inclure une fin , lorsque vous spécifiez un seul plugin. Par exemple, `plugins = ('EC2Plugin',)`

Ajout des intergiciels à votre application (Flask)

Pour instrumenter l'application Flask, vous devez d'abord configurer un nom de segment dans l'élément `xray_recorder`. Utilisez ensuite la fonction `XRayMiddleware` pour corriger votre application Flask dans le code.

Exemple `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)
```

Cela indique à l'enregistreur X-Ray de suivre les demandes traitées par votre application Flask avec le taux d'échantillonnage par défaut. Vous pouvez [configurer l'enregistreur dans le code](#) pour appliquer les règles d'échantillonnage personnalisées ou modifier d'autres paramètres.

Ajout des intergiciels à votre application (Bottle)

Pour instrumenter votre application Bottle, vous devez d'abord configurer un nom de segment dans l'élément `xray_recorder`. Utilisez ensuite la fonction `XRayMiddleware` pour corriger votre application Bottle dans le code.

Exemple `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))
```

Cela indique à l'enregistreur X-Ray de suivre les demandes traitées par votre application Bottle avec le taux d'échantillonnage par défaut. Vous pouvez [configurer l'enregistreur dans le code](#) pour appliquer les règles d'échantillonnage personnalisées ou modifier d'autres paramètres.

Instrumentation manuelle du code Python

Si vous n'utilisez pas Django ni Flask, vous pouvez créer des segments manuellement. Vous pouvez créer un segment pour chaque demande entrante, ou créer des segments autour de clients HTTP ou AWS SDK corrigés afin de fournir un contexte permettant à l'enregistreur d'ajouter des sous-segments.

Exemple main.py — Instrumentation manuelle

```
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()
```

Configuration d'une stratégie d'attribution de noms de segment

AWS X-Ray utilise un nom de service pour identifier votre application et la distinguer des autres applications, bases de données, ressources externes APIs et AWS ressources utilisées par votre application. Lorsque le SDK X-Ray génère des segments pour les demandes entrantes, il enregistre le nom du service de votre application dans le [champ du nom](#) du segment.

Le SDK X-Ray peut nommer les segments d'après le nom d'hôte dans l'en-tête de la requête HTTP. Cependant, cet en-tête peut être falsifié, ce qui peut entraîner la création de nœuds inattendus dans votre carte de service. Pour éviter que le SDK ne nomme les segments de manière incorrecte en raison de demandes contenant des en-têtes d'hôte falsifiés, vous devez spécifier un nom par défaut pour les demandes entrantes.

Si votre application traite des demandes pour plusieurs domaines, vous pouvez configurer le SDK pour qu'il utilise une stratégie de dénomination dynamique afin de refléter cela dans les noms des segments. Une stratégie de dénomination dynamique permet au SDK d'utiliser le nom d'hôte pour les demandes qui correspondent à un modèle attendu et d'appliquer le nom par défaut aux demandes qui ne le sont pas.

Par exemple, il se peut qu'une seule application envoie des demandes à trois sous-domaines : `www.example.com`, `api.example.com`, `etstatic.example.com`. Vous pouvez utiliser une stratégie de dénomination dynamique avec le modèle `*.example.com` pour identifier les segments de chaque sous-domaine avec un nom différent, ce qui permet d'obtenir trois nœuds de service sur la carte des services. Si votre application reçoit des demandes dont le nom d'hôte ne correspond pas au modèle, vous verrez un quatrième nœud sur la carte des services avec un nom de remplacement que vous spécifiez.

Pour utiliser le même nom pour tous les segments de demande, spécifiez le nom de votre application lorsque vous configurez l'enregistreur, comme illustré dans les [sections précédentes](#).

Une stratégie d'attribution de noms dynamique définit un modèle auquel doivent correspondre les noms d'hôte et un nom par défaut à utiliser si le nom d'hôte de la demande HTTP ne correspond pas au modèle. Pour nommer des segments de façon dynamique dans Django, ajoutez le paramètre `DYNAMIC_NAMING` au fichier [settings.py](#).

Exemple settings.py — Dénomination dynamique

```
XRAY_RECORDER = {
    'AUTO_INSTRUMENT': True,
    'AWS_XRAY_TRACING_NAME': 'My application',
    'DYNAMIC_NAMING': '*.example.com',
    'PLUGINS': ('ElasticBeanstalkPlugin', 'EC2Plugin')
}
```

Vous pouvez utiliser `*` dans le modèle pour correspondre à n'importe quelle chaîne ou `?` pour correspondre à un seul caractère. Pour Flask, [configurez l'enregistreur dans le code](#).

Exemple main.py — Nom du segment

```
from aws_xray_sdk.core import xray_recorder
xray_recorder.configure(service='My application')
xray_recorder.configure(dynamic_naming='*.example.com')
```

Note

Vous pouvez remplacer le nom de service par défaut que vous définissez avec la `AWS_XRAY_TRACING_NAME` variable d'environnement [???](#).

Correction des bibliothèques pour instrumenter les appels en aval

Pour instrumenter les appels en aval, utilisez le SDK X-Ray pour Python afin de patcher les bibliothèques utilisées par votre application. Le SDK X-Ray pour Python peut appliquer des correctifs aux bibliothèques suivantes.

Bibliothèques prises en charge

- [botocore](#), [boto3](#) — AWS SDK for Python (Boto) Clients de l'instrument.
- [pynamodb](#) — Version du client Amazon DynamoDB de l'instrument PynamoDB.
- [aiobotocore](#), [aioboto3](#) — Versions intégrées à Instrument [asyncio](#) du SDK pour les clients Python.
- [requests](#), [aiohttp](#) — Clients HTTP de haut niveau de l'instrument.
- [httplib](#), [http.client](#) — Instrument les clients HTTP de bas niveau et les bibliothèques de niveau supérieur qui les utilisent.
- [sqlite3](#) — SQLite Clients de l'instrument.
- [mysql-connector-python](#) — Instrument les clients MySQL.
- [pg8000](#) — Interface PostgreSQL en Python pur de l'instrument.
- [psycopg2](#) — Adaptateur de base de données PostgreSQL Instrument.
- [pymongo](#) — Instrument les clients MongoDB.
- [pymysql](#) — Clients basés sur Instrument PyMy SQL pour MySQL et MariaDB.

Lorsque vous utilisez une bibliothèque patchée, le SDK X-Ray pour Python crée un sous-segment pour l'appel et enregistre les informations de la demande et de la réponse. Un segment doit être

disponible pour que le kit SDK puisse créer le sous-segment à partir de l'intergiciel SDK ou d' AWS Lambda.

Note

Si vous utilisez SQLAlchemy ORM, vous pouvez instrumenter vos requêtes SQL en important la version du SDK des classes SQLAlchemy de session et de requête. Voir [Utiliser l' SQLAlchemyORM](#) pour obtenir des instructions.

Pour corriger toutes les bibliothèques disponibles, utilisez la fonction `patch_all` dans `aws_xray_sdk.core`. Certaines bibliothèques, telles que `httplib` et `urllib`, peuvent avoir besoin d'activer la double application de correctifs en appelant `patch_all(double_patch=True)`.

Exemple `main.py` — Corrige toutes les bibliothèques prises en charge

```
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()
```

Pour corriger une seule bibliothèque, appelez le `patch` selon un tuple du nom de la bibliothèque. Pour ce faire, vous devrez fournir une liste d'éléments unique.

Exemple `main.py` — Bibliothèques spécifiques aux correctifs

```
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

Dans certains cas, la clé que vous utilisez pour corriger une bibliothèque ne correspond pas au nom de la bibliothèque. Certaines clés tiennent lieu d'alias pour une ou plusieurs bibliothèques.

Alias de bibliothèques

- `httplib` – [httplib](#) et [http.client](#)
- `mysql` – [mysql-connector-python](#)

Suivi du contexte pour le travail asynchrone

Pour les bibliothèques `asyncio` intégrées ou pour [créer des sous-segments pour les fonctions asynchrones](#), vous devez également configurer le SDK X-Ray pour Python avec un contexte asynchrone. Importez la `AsyncContext` classe et transmettez-en une instance à l'enregistreur X-Ray.

 Note

Les bibliothèques de support de l'infrastructure web, comme `AIOHTTP`, ne sont pas traitées via le module `aws_xray_sdk.core.patcher`. Elles n'apparaîtront pas dans le catalogue `patcher` des bibliothèques prises en charge.

Exemple `main.py` — Correctif `aioboto3`

```
import asyncio
import aioboto3
import requests

from aws_xray_sdk.core.async_context import AsyncContext
from aws_xray_sdk.core import xray_recorder
xray_recorder.configure(service='my_service', context=AsyncContext())
from aws_xray_sdk.core import patch

libraries = (['aioboto3'])
patch(libraries)
```

Suivi des appels du AWS SDK avec le SDK X-Ray pour Python

Lorsque votre application fait des appels Services AWS pour stocker des données, écrire dans une file d'attente ou envoyer des notifications, le SDK X-Ray pour Python suit les appels en aval dans des [sous-segments](#). Les ressources tracées Services AWS et auxquelles vous accédez au sein de ces services (par exemple, un compartiment Amazon S3 ou une file d'attente Amazon SQS) apparaissent sous forme de nœuds en aval sur la carte de suivi de la console X-Ray.

Le SDK X-Ray pour Python instrumente automatiquement tous les clients du AWS SDK lorsque vous appliquez un [correctif à la botocore bibliothèque](#). Vous ne pouvez pas instrumenter des clients individuels.

Pour tous les services, vous pouvez voir le nom de l'API appelée dans la console X-Ray. Pour un sous-ensemble de services, le SDK X-Ray ajoute des informations au segment afin d'améliorer la granularité de la carte des services.

Par exemple, lorsque vous passez un appel avec un client DynamoDB instrumenté, le SDK ajoute le nom de la table au segment pour les appels qui ciblent une table. Dans la console, chaque table apparaît sous la forme d'un nœud distinct dans la carte des services, avec un nœud DynamoDB générique pour les appels qui ne ciblent pas une table.

Exemple Sous-segment pour un appel à DynamoDB pour enregistrer un élément

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

Lorsque vous accédez aux ressources nommées, les appels vers les services suivants créent des nœuds supplémentaires dans la cartographie des services. Les appels qui ne ciblent pas de ressources spécifiques créent un nœud générique pour le service.

- Amazon DynamoDB — Nom de la table
- Amazon Simple Storage Service : nom du compartiment et de la clé
- Amazon Simple Queue Service — Nom de la file d'attente

Suivi des appels aux services Web HTTP en aval à l'aide du SDK X-Ray pour Python

Lorsque votre application appelle des microservices ou du protocole HTTP public APIs, vous pouvez utiliser le SDK X-Ray pour Python pour instrumenter ces appels et ajouter l'API au graphe de service en tant que service en aval.

Pour instrumenter les clients HTTP, [corrigez la bibliothèque](#) que vous utilisez pour effectuer les appels sortants. Si vous utilisez `requests` ou Python intégré dans un client HTTP, c'est tout ce dont vous avez besoin de faire. Pour `aiohttp`, configurez également l'enregistreur avec un [contexte asynchrone](#).

Si vous utilisez l'API client de `aiohttp` 3, vous devez également configurer `ClientSession` avec une instance de la configuration de suivi fournie par le kit SDK.

Exemple [API Client aiohttp 3](#)

```
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()
```

Lorsque vous instrumentez un appel à une API Web en aval, le SDK X-Ray pour Python enregistre un sous-segment contenant des informations sur la requête et la réponse HTTP. X-Ray utilise le sous-segment pour générer un segment inféré pour l'API distante.

Exemple Sous-segment pour un appel HTTP en aval

```
{
  "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
    }
  }
}
```

Exemple Segment déduit pour un appel HTTP en aval

```
{
  "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
}
```

Génération de sous-segments personnalisés avec le SDK X-Ray pour Python

Les sous-segments étendent le [segment](#) d'une trace avec des détails sur le travail effectué afin de répondre à une demande. Chaque fois que vous passez un appel avec un client instrumenté, le SDK X-Ray enregistre les informations générées dans un sous-segment. Vous pouvez créer des sous-segments supplémentaires pour regrouper d'autres sous-segments, pour mesurer les performances d'une section de code ou pour enregistrer des annotations et des métadonnées.

Pour gérer des sous-segments, utilisez les méthodes `begin_subsegment` et `end_subsegment`.

Exemple `main.py` — Sous-segment personnalisé

```
from aws_xray_sdk.core import xray_recorder

subsegment = xray_recorder.begin_subsegment('annotations')
subsegment.put_annotation('id', 12345)
xray_recorder.end_subsegment()
```

Pour créer un sous-segment pour une fonction synchrone, utilisez le décorateur `@xray_recorder.capture`. Vous pouvez transmettre un nom pour le sous-segment à la fonction de capture ou le laisser utiliser le nom de la fonction.

Exemple `main.py` — Sous-segment de fonction

```
from aws_xray_sdk.core import xray_recorder

@xray_recorder.capture('## create_user')
def create_user():
    ...
```

Pour une fonction asynchrone, utilisez le décorateur `@xray_recorder.capture_async` et transmettez un contexte asynchrone à l'enregistreur.

Exemple `main.py` — Sous-segment de fonction asynchrone

```
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()
```

Lorsque vous créez un sous-segment au sein d'un segment ou d'un autre sous-segment, le SDK X-Ray pour Python génère un identifiant et enregistre l'heure de début et de fin.

Exemple Sous-segment avec des métadonnées

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

Ajoutez des annotations et des métadonnées aux segments avec le SDK X-Ray pour Python

Vous pouvez enregistrer des informations supplémentaires sur les demandes, l'environnement ou votre application à l'aide d'annotations et de métadonnées. Vous pouvez ajouter des annotations et des métadonnées aux segments créés par le SDK X-Ray ou aux sous-segments personnalisés que vous créez.

Les annotations sont des paires clé-valeur avec des chaînes, des nombres ou des valeurs booléennes. Les annotations sont indexées pour être utilisées avec les expressions de [filtre](#). Utilisez les annotations pour enregistrer les données que vous souhaitez utiliser pour regrouper les suivis dans la console ou lors de l'appel de l'API [GetTraceSummaries](#).

Les métadonnées sont des paires clé-valeur qui peuvent contenir des valeurs de n'importe quel type, y compris des objets et des listes, mais qui ne sont pas indexées pour être utilisées avec des expressions de filtre. Utilisez les métadonnées pour enregistrer des données supplémentaires

que vous souhaitez stocker dans le traçage, mais dont vous n'avez pas besoin pour effectuer une recherche.

En plus des annotations et des métadonnées, vous pouvez également [enregistrer les chaînes d'ID utilisateur](#) sur des segments. Les utilisateurs sont enregistrés dans un champ distinct sur les segments et sont indexés pour être utilisés dans le cadre de la recherche.

Sections

- [Enregistrer des annotations avec le SDK X-Ray pour Python](#)
- [Enregistrement de métadonnées avec le SDK X-Ray pour Python](#)
- [Utilisateur enregistrant IDs avec le SDK X-Ray pour Python](#)

Enregistrer des annotations avec le SDK X-Ray pour Python

Utilisez les annotations pour enregistrer les informations sur les segments ou sous-segments qui doivent être indexés pour la recherche.

Exigences liées aux annotations

- Clés — La clé d'une annotation X-Ray peut comporter jusqu'à 500 caractères alphanumériques. Vous ne pouvez pas utiliser d'espaces ou de symboles autres qu'un point ou un point (.)
- Valeurs — La valeur d'une annotation X-Ray peut comporter jusqu'à 1 000 caractères Unicode.
- Nombre d'annotations : vous pouvez utiliser jusqu'à 50 annotations par trace.

Pour enregistrer les annotations

1. Obtenez une référence au segment ou sous-segment en cours à partir d'`xray_recorder`.

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

or

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


2. Appelez `put_annotation` avec une clé de type chaîne et une valeur de type booléen, numérique ou chaîne.

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

L'exemple suivant montre comment appeler `putAnnotation` avec une clé `String` qui inclut un point et une valeur booléenne, numérique ou chaîne.

```
document.putAnnotation("testkey.test", "my value");
```

Vous pouvez également utiliser la méthode `put_annotation` sur le `xray_recorder`. Cette méthode enregistre les annotations sur le sous-segment actuel ou, si aucun sous-segment n'est ouvert, sur le segment.

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

Le kit de développement logiciel enregistre les annotations sous forme de paires clé-valeur dans un objet `annotations` du document de segment. Si vous appelez deux fois `put_annotation` avec la même clé, les valeurs précédemment enregistrées sur le même segment ou sous-segment sont remplacées.

Pour rechercher les suivis ayant des annotations avec des valeurs spécifiques, utilisez le mot clé `annotation[key]` dans une [expression de filtre](#).

Enregistrement de métadonnées avec le SDK X-Ray pour Python

Utilisez les métadonnées pour enregistrer des informations sur les segments ou sous-segments qui n'ont pas besoin d'être indexés pour la recherche. Les valeurs des métadonnées peuvent être de type chaîne, valeur numérique, valeur booléenne ou tout objet à même d'être sérialisé en un tableau ou objet JSON.

Pour enregistrer les métadonnées

1. Obtenez une référence au segment ou sous-segment en cours à partir d'`xray_recorder`.

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

or

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

2. Appelez `put_metadata` avec une clé de type chaîne, une valeur de type booléen, numérique, chaîne ou objet et un espace de noms de type chaîne.

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

or

Appelez `put_metadata` avec seulement une clé et une valeur.

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

Vous pouvez également utiliser la méthode `put_metadata` sur le `xray_recorder`. Cette méthode enregistre les métadonnées sur le sous-segment actuel ou, si aucun sous-segment n'est ouvert, sur le segment.

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

Si vous ne spécifiez pas d'espace de noms, le kit de développement logiciel utilise `default`. Si vous appelez deux fois `put_metadata` avec la même clé, les valeurs précédemment enregistrées sur le même segment ou sous-segment sont remplacées.

Utilisateur enregistrant IDs avec le SDK X-Ray pour Python

Enregistrez les segments utilisateur IDs sur demande pour identifier l'utilisateur qui a envoyé la demande.

Pour enregistrer un utilisateur IDs

1. Obtenez une référence au segment en cours à partir d'`xray_recorder`.

```
from aws_xray_sdk.core import xray_recorder
...
```

```
document = xray_recorder.current_segment()
```

2. Appelez `setUser` avec l'ID de type chaîne de l'utilisateur ayant envoyé la demande.

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

Vous pouvez appeler `set_user` dans vos contrôleurs pour enregistrer l'ID utilisateur dès que votre application commence le traitement d'une demande.

Pour rechercher les suivis d'un ID utilisateur, utilisez le mot clé `user` dans une [expression de filtre](#).

Instrumentation des infrastructures web déployées dans des environnements sans serveur

Le SDK AWS X-Ray pour Python permet d'instrumenter les frameworks Web déployés dans des applications sans serveur. Le sans serveur désigne l'architecture native du cloud qui vous permet de déléguer davantage de responsabilités opérationnelles à AWS ce qui augmente votre agilité et votre innovation.

L'architecture sans serveur est un modèle d'application logicielle qui vous permet de créer et d'exécuter des applications et des services sans vous soucier des serveurs. Cela permet d'éliminer les tâches de gestion des infrastructures, comme le provisionnement de serveur ou de cluster, la correction, la maintenance des systèmes d'exploitation et l'allocation de capacités. Vous pouvez en concevoir des solutions sans serveur pour pratiquement n'importe quel type d'application ou de service backend. Tous les aspects nécessaires à l'exécution et à la mise à l'échelle de votre application avec une haute disponibilité sont gérés à votre place.

Ce didacticiel explique comment AWS X-Ray instrumenter automatiquement un framework Web, tel que Flask ou Django, déployé dans un environnement sans serveur. L'instrumentation X-Ray de l'application vous permet de visualiser tous les appels en aval effectués, en commençant par Amazon API Gateway via votre AWS Lambda fonction, et les appels sortants effectués par votre application.

Le SDK X-Ray pour Python prend en charge les frameworks d'applications Python suivants :

- Flask version 0.8 ou ultérieure
- Django version 1.0 ou ultérieure

Ce didacticiel développe un exemple d'application sans serveur déployée sur Lambda et invoquée par API Gateway. Ce didacticiel utilise Zappa pour déployer automatiquement l'application sur Lambda et pour configurer le point de terminaison API Gateway.

Prérequis

- [Zappa](#)
- [Python](#) — Version 2.7 ou 3.6.
- [AWS CLI](#)— Vérifiez que vous êtes AWS CLI configuré avec le compte Région AWS dans lequel vous allez déployer votre application.
- [Pip](#)
- [Virtualenv](#)

Étape 1 : Créer un environnement

Au cours de cette étape, vous allez créer un environnement virtuel à l'aide de `virtualenv` pour héberger une application.

1. À l'aide du AWS CLI, créez un répertoire pour l'application. Accédez ensuite au nouveau répertoire.

```
mkdir serverless_application  
cd serverless_application
```

2. Créez ensuite un environnement virtuel au sein de votre nouveau répertoire. Utilisez la commande suivante pour l'activer.

```
# Create our virtual environment  
virtualenv serverless_env  
  
# Activate it  
source serverless_env/bin/activate
```

3. Installez X-Ray, Flask, Zappa et la bibliothèque Requests dans votre environnement.

```
# Install X-Ray, Flask, Zappa, and Requests into your environment  
pip install aws-xray-sdk flask zappa requests
```

4. Ajoutez le code d'application au répertoire `serverless_application`. Pour cet exemple, nous pouvons effectuer la création à partir de l'exemple [Hello World](#) de Flask.

Dans le répertoire `serverless_application`, créez un fichier nommé `my_app.py`. Utilisez ensuite un éditeur de texte pour ajouter les commandes suivantes. Cette application instrumente la bibliothèque Requests, applique les correctifs de l'intergiciel de l'application Flask et ouvre le point de terminaison `'/'`.

```
# 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
```

Étape 2 : Créer et déployer un environnement Zappa

Au cours de cette étape, vous utiliserez Zappa pour configurer automatiquement un point de terminaison API Gateway, puis le déployer sur Lambda.

1. Initialisez Zappa depuis le répertoire `serverless_application`. Pour cet exemple, nous utilisons les paramètres par défaut, mais si vous avez des préférences de personnalisation, Zappa affiche des instructions de configuration.

```
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. Activez X-Ray. Ouvrez le fichier `zappa_settings.json` et vérifiez s'il ressemble à l'exemple.

```

{
  "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. Ajoutez `"xray_tracing": true` comme entrée au fichier de configuration.

```

{
  "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. Déployez l'application. Cela configure automatiquement le point de terminaison API Gateway et télécharge votre code sur Lambda.

```
zappa deploy
```

```
...  
Deploying API Gateway..  
Deployment complete!: https://*****.execute-api.us-west-2.amazonaws.com/dev
```

Étape 3 : activer le suivi X-Ray pour API Gateway

Au cours de cette étape, vous allez interagir avec la console API Gateway pour activer le suivi X-Ray.

1. Connectez-vous à la console API Gateway AWS Management Console et ouvrez-la à l'adresse <https://console.aws.amazon.com/apigateway/>.
2. Recherchez votre API nouvellement générée. Vous devez obtenir quelque chose de similaire à `serverless-exam-dev`.
3. Choisissez Stages (Étapes).
4. Choisissez le nom de votre étape de déploiement. L'argument par défaut est `dev`.
5. Sous l'onglet Logs/Tracing (Journaux/Suivi), cochez la case Enable X-Ray Tracing (Activer le suivi X-Ray).
6. Choisissez Save Changes (Enregistrer les modifications).
7. Accédez au point de terminaison dans votre navigateur. Si vous avez utilisé l'exemple d'application Hello World, il doit afficher les éléments suivants.

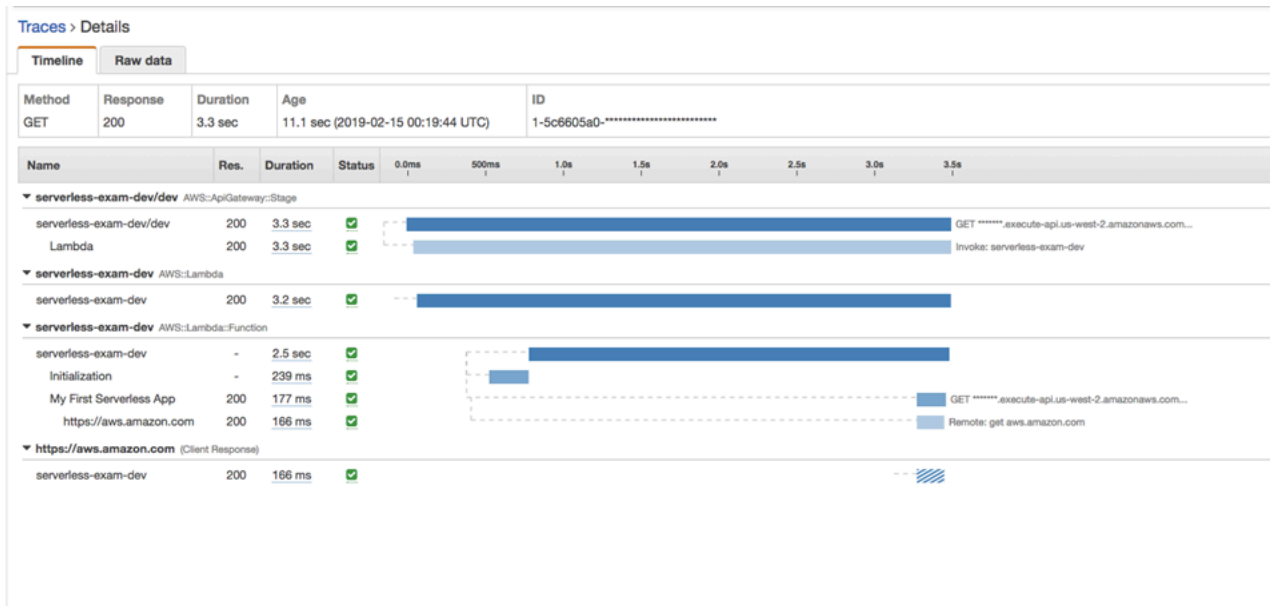
```
"Hello, World: https://aws.amazon.com/"
```

Étape 4 : Afficher le suivi créé

Au cours de cette étape, vous allez interagir avec la console X-Ray pour afficher la trace créée par l'exemple d'application. Pour obtenir une présentation plus détaillée de l'analyse des suivis, veuillez consulter [Affichage de la cartographie des services](#).

1. Connectez-vous à la console X-Ray AWS Management Console et ouvrez-la <https://console.aws.amazon.com/xray/chez> vous.

2. Affichez les segments générés par API Gateway, la fonction Lambda et le conteneur Lambda.
3. Sous le segment de fonction Lambda, visualisez un sous-segment nommé `My First Serverless App` Il est suivi par un deuxième sous-segment nommé `https://aws.amazon.com`.
4. Lors de l'initialisation, Lambda peut également générer un troisième sous-segment nommé `initialization`



Étape 5 : nettoyer

Supprimez toujours les ressources que vous n'utilisez plus pour éviter d'accumuler des coûts imprévus. Comme le montre ce tutoriel, des outils tels que Zappa rationalisent le redéploiement sans serveur.

Pour supprimer votre application de Lambda, d'API Gateway et d'Amazon S3, exécutez la commande suivante dans le répertoire de votre projet à l'aide du AWS CLI

```
zappa undeploy dev
```

Étapes suivantes

Ajoutez des fonctionnalités supplémentaires à votre application en ajoutant des AWS clients et en les instrumentant avec X-Ray. Pour en savoir plus sur les options informatiques sans serveur, rendez-vous sur [Serverless](#) on. AWS

Utilisation de .NET

Il existe deux manières d'instrumenter votre application .NET pour envoyer des traces à X-Ray :

- [AWS Distro pour OpenTelemetry .NET](#) : AWS distribution qui fournit un ensemble de bibliothèques open source permettant d'envoyer des métriques et des traces corrélées à plusieurs solutions de AWS surveillance CloudWatch, notamment Amazon et Amazon OpenSearch Service AWS X-Ray, via [AWS Distro](#) for Collector. OpenTelemetry
- [AWS X-Ray SDK pour .NET](#) : ensemble de bibliothèques permettant de générer et d'envoyer des traces à X-Ray via le daemon [X-Ray](#).

Pour de plus amples informations, veuillez consulter [Choisir entre The AWS Distro for OpenTelemetry et X-Ray SDKs](#).

AWS Distro pour .NET OpenTelemetry

Avec le AWS Distro pour OpenTelemetry .NET, vous pouvez instrumenter vos applications une seule fois et envoyer des métriques et des traces corrélées à plusieurs solutions de AWS surveillance CloudWatch AWS X-Ray, notamment Amazon et Amazon OpenSearch Service. L'utilisation de X-Ray avec AWS Distro pour OpenTelemetry nécessite deux composants : un OpenTelemetry SDK activé pour une utilisation avec X-Ray, et un AWS Distro for OpenTelemetry Collector activé pour une utilisation avec X-Ray.

Pour commencer, consultez la [documentation de AWS Distro pour OpenTelemetry .NET](#).

Pour plus d'informations sur l'utilisation de la AWS distribution pour OpenTelemetry with AWS X-Ray et d'autres Services AWS, consultez la section [AWS Distro for OpenTelemetry](#) ou la distribution pour la [AWS documentation](#). OpenTelemetry

Pour plus d'informations sur la prise en charge et l'utilisation des langues, consultez [AWS Observability on GitHub](#).

AWS X-Ray SDK pour .NET

Le SDK X-Ray pour .NET est une bibliothèque permettant d'instrumenter les applications Web .NET C#, les applications Web .NET Core et les fonctions .NET Core sur AWS Lambda Il fournit des classes et des méthodes pour générer et envoyer des données de trace au [daemon X-Ray](#). Cela

inclut des informations sur les demandes entrantes traitées par l'application et les appels que l'application effectue vers les bases de données en aval Services AWS, le Web APIs HTTP et les bases de données SQL.

Note

Le SDK X-Ray pour .NET est un projet open source. Vous pouvez suivre le projet et soumettre des problèmes et des pull requests sur GitHub : github.com/aws/aws-xray-sdk-dotnet

Pour les applications web, commencez par [ajouter un gestionnaire de messages à votre configuration web](#) pour suivre les demandes entrantes. Le gestionnaire de messages crée un [segment](#) pour chaque demande suivie et met fin au segment lorsque la réponse est envoyée. Tandis que le segment soit ouvert, vous pouvez utiliser les méthodes du client du SDK pour ajouter des informations au segment et créer des sous-segments afin de suivre les appels en amont. Le kit de développement logiciel (SDK) enregistre aussi automatiquement les exceptions que votre application renvoie quand le segment est ouvert.

Pour les fonctions Lambda appelées par une application ou un service instrumenté, Lambda lit l'[en-tête de suivi et trace automatiquement les requêtes échantillonnées](#). Pour les autres fonctions, vous pouvez [configurer Lambda](#) pour échantillonner et suivre les demandes entrantes. Dans les deux cas, Lambda crée le segment et le fournit au SDK X-Ray.

Note

Sur Lambda, le SDK X-Ray est facultatif. Si vous ne l'utilisez pas dans votre fonction, votre carte de service inclura toujours un nœud pour le service Lambda et un nœud pour chaque fonction Lambda. En ajoutant le SDK, vous pouvez instrumenter votre code de fonction pour ajouter des sous-segments au segment de fonction enregistré par Lambda. Pour plus d'informations, consultez [AWS Lambda et AWS X-Ray](#).

Utilisez ensuite le SDK X-Ray pour .NET pour [instrumenter AWS SDK pour .NET vos clients](#). Chaque fois que vous appelez une ressource Service AWS ou une ressource en aval avec un client instrumenté, le SDK enregistre les informations relatives à l'appel dans un sous-segment. AWS les services et les ressources auxquels vous accédez au sein des services apparaissent sous forme

de nœuds en aval sur la carte de trace pour vous aider à identifier les erreurs et les problèmes de limitation sur les connexions individuelles.

Le SDK X-Ray pour .NET fournit également des instruments pour les appels en aval [aux bases de données APIs Web HTTP et SQL](#). La méthode d'extension `GetResponseTraced` pour `System.Net.HttpWebRequest` suit les appels HTTP sortants. Vous pouvez utiliser la version du SDK X-Ray pour .NET pour `SqLCommand` instrumenter les requêtes SQL.

Après avoir commencé à utiliser le SDK, personnalisez son comportement en [configurant l'enregistreur et le gestionnaire de messages](#). Vous pouvez ajouter des plug-ins pour enregistrer les données sur les ressources de calcul exécutant votre application, personnaliser le comportement d'échantillonnage en définissant des règles d'échantillonnage et définir le niveau de journal afin d'afficher plus ou moins d'informations du kit de développement logiciel dans vos journaux d'application.

Enregistrez les informations supplémentaires sur les demandes et le travail que votre application effectue dans les [annotations et les métadonnées](#). Les annotations sont de simples paires clé-valeur, indexées en vue de leur utilisation avec les [expressions de filtre](#), de telle sorte que vous pouvez explorer les suivis qui contiennent des données spécifiques. Les entrées des métadonnées sont moins restrictives et peuvent enregistrer des objets et tableaux entiers (tout ce qui peut être sérialisé en JSON).

Annotations et métadonnées

Les annotations et les métadonnées sont du texte arbitraire que vous ajoutez aux segments avec le SDK X-Ray. Les annotations sont indexées pour être utilisées avec les expressions de filtre. Les métadonnées ne sont pas indexées, mais peuvent être consultées dans le segment brut à l'aide de la console ou de l'API X-Ray. Toute personne à qui vous accordez un accès en lecture à X-Ray peut consulter ces données.

Lorsque vous avez un grand nombre de clients instrumentés dans votre code, un seul segment de demande peut contenir un grand nombre de sous-segments, un par appel effectué à l'aide d'un client instrumenté. Vous pouvez organiser et grouper les sous-segments en enveloppant les appels clients dans des [sous-segments personnalisés](#). Vous pouvez créer un sous-segment personnalisé pour une fonction complète ou une quelconque section du code, puis enregistrer les métadonnées et les annotations sur le sous-segment au lieu de tout écrire sur le segment parent.

Pour obtenir de la documentation de référence sur les classes et méthodes du kit de développement logiciel, consultez les ressources suivantes :

- [AWS X-Ray Référence de l'API SDK for .NET](#)
- [AWS X-Ray Référence de l'API SDK for .NET Core](#)

Le même package prend en charge à la fois .NET et .NET Core, mais les classes utilisées varient. Les exemples contenus dans ce chapitre sont liés à la référence d'API .NET, sauf si la classe est propre à .NET Core.

Prérequis

Le SDK X-Ray pour .NET nécessite le .NET Framework 4.5 ou version ultérieure AWS SDK pour .NET et.

Pour les applications et fonctions .NET Core, le kit SDK requiert .NET Core 2.0 ou une version ultérieure.

Ajout du SDK X-Ray pour .NET à votre application

NuGet À utiliser pour ajouter le SDK X-Ray pour .NET à votre application.

Pour installer le SDK X-Ray pour .NET NuGet avec le gestionnaire de packages dans Visual Studio

1. Choisissez Outils, Gestionnaire de NuGet packages, Gérer les NuGet packages pour la solution.
2. Recherchez AWSXRayRecorder.
3. Choisissez le package, puis choisissez Install.

Gestion des dépendances

Le SDK X-Ray pour .NET est disponible [auprès de](#) Nuget. Installez le SDK à l'aide du gestionnaire de packages :

```
Install-Package AWSXRayRecorder -Version 2.10.1
```

Le package AWSXRayRecorder v2.10.1 nuget possède les dépendances suivantes :

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)
```

Pour plus de détails sur la gestion des dépendances, consultez la documentation Microsoft sur la [dépendance Nuget et la résolution des dépendances Nuget](#).

Configuration du kit de développement X-Ray pour .NET

Vous pouvez configurer le SDK X-Ray pour .NET à l'aide de plug-ins afin d'inclure des informations sur le service sur lequel votre application s'exécute, de modifier le comportement d'échantillonnage par défaut ou d'ajouter des règles d'échantillonnage qui s'appliquent aux demandes relatives à des chemins spécifiques.

Pour les applications web .NET, ajoutez des clés à la section appSettings du fichier Web.config.

Exemple Web.config

```
<configuration>
  <appSettings>
    <add key="AWSXRayPlugins" value="EC2Plugin"/>
    <add key="SamplingRuleManifest" value="sampling-rules.json"/>
  </appSettings>
</configuration>
```

Pour .NET Core, créez un fichier nommé appsettings.json avec une clé de niveau supérieur nommée XRay.

Exemple .NET appsettings.json

```
{
  "XRay": {
    "AWSXRayPlugins": "EC2Plugin",
```

```
"SamplingRuleManifest": "sampling-rules.json"
}
}
```

Ensuite, dans le code de votre application, créez un objet de configuration et utilisez-le pour initialiser l'enregistreur X-Ray. Faites-le avant d'[initialiser l'enregistreur](#).

Exemple .NET Core Program.cs — Configuration de l'enregistreur

```
using Amazon.XRay.Recorder.Core;
...
AWSXRayRecorder.InitializeInstance(configuration);
```

Si vous instrumentez une application web .NET Core, vous pouvez également transmettre l'objet de configuration à la méthode UseXRay lorsque vous [configurez le gestionnaire de messages](#). Pour les fonctions Lambda, utilisez la InitializeInstance méthode décrite ci-dessus.

Pour plus d'informations sur l'API de configuration .NET Core, voir [Configurer une application ASP.NET Core](#) sur docs.microsoft.com.

Sections

- [Plugins](#)
- [Règles d'échantillonnage](#)
- [Journalisation \(.NET\)](#)
- [Journalisation \(.NET Core\)](#)
- [Variables d'environnement](#)

Plugins

Utilisez des plug-ins pour ajouter des données sur le service qui héberge votre application.

Plugins

- Amazon EC2 — EC2Plugin ajoute l'ID de l'instance, la zone de disponibilité et le groupe de CloudWatch journaux.
- Elastic BeanstalkPlugin Beanstalk : ajoute le nom de l'environnement, l'étiquette de version et l'ID de déploiement.
- Amazon ECS — ECSPlugin ajoute l'ID du conteneur.

Pour utiliser un plugin, configurez le client X-Ray SDK for .NET en ajoutant `AWSXRayPlugins` le paramètre. Si plusieurs plug-ins s'appliquent à votre application, spécifiez-les tous dans le même paramètre, séparés par des virgules.

Exemple Web.config - plug-ins

```
<configuration>
  <appSettings>
    <add key="AWSXRayPlugins" value="EC2Plugin,ElasticBeanstalkPlugin"/>
  </appSettings>
</configuration>
```

Exemple .NET Core appsettings.json — Plug-ins

```
{
  "XRay": {
    "AWSXRayPlugins": "EC2Plugin,ElasticBeanstalkPlugin"
  }
}
```

Règles d'échantillonnage

Le SDK utilise les règles d'échantillonnage que vous définissez dans la console X-Ray pour déterminer les demandes à enregistrer. La règle par défaut suit la première demande chaque seconde, et 5 % de toutes les demandes supplémentaires provenant de tous les services envoient des traces à X-Ray. [Créez des règles supplémentaires dans la console X-Ray](#) pour personnaliser la quantité de données enregistrées pour chacune de vos applications.

Le SDK applique les règles personnalisées dans l'ordre dans lequel elles sont définies. Si une demande correspond à plusieurs règles personnalisées, le SDK applique uniquement la première règle.

Note

Si le SDK ne parvient pas à accéder à X-Ray pour obtenir des règles d'échantillonnage, il revient à une règle locale par défaut concernant la première demande chaque seconde, et 5 % de toutes les demandes supplémentaires par hôte. Cela peut se produire si l'hôte n'est pas autorisé à appeler `sampling APIs` ou ne peut pas se connecter au daemon X-Ray, qui agit comme un proxy TCP pour les appels d'API effectués par le SDK.

Vous pouvez également configurer le SDK pour charger des règles d'échantillonnage à partir d'un document JSON. Le SDK peut utiliser les règles locales comme solution de rechange dans les cas où l'échantillonnage X-Ray n'est pas disponible, ou utiliser exclusivement les règles locales.

Exemple `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
  }
}
```

Cet exemple définit une règle personnalisée et une règle par défaut. La règle personnalisée applique un taux d'échantillonnage de 5 % sans nombre minimum de demandes à suivre pour les chemins sous-jacents. `/api/move/` La règle par défaut suit la première demande chaque seconde et 10 % des demandes supplémentaires.

L'inconvénient de définir des règles localement est que la cible fixe est appliquée par chaque instance de l'enregistreur indépendamment, au lieu d'être gérée par le service X-Ray. Au fur et à mesure que vous déployez de nouveaux hôtes, le taux fixe est multiplié, ce qui complique le contrôle de la quantité de données enregistrées.

Activé AWS Lambda, vous ne pouvez pas modifier le taux d'échantillonnage. Si votre fonction est appelée par un service instrumenté, les appels ayant généré des demandes échantillonnées par ce service seront enregistrés par Lambda. Si le suivi actif est activé et qu'aucun en-tête de suivi n'est présent, Lambda prend la décision d'échantillonnage.

Pour configurer les règles de sauvegarde, demandez au SDK X-Ray pour .NET de charger les règles d'échantillonnage à partir d'un fichier contenant `SamplingRuleManifest` le paramètre.

Exemple .NET Web.config - Règles d'échantillonnage

```
<configuration>
  <appSettings>
    <add key="SamplingRuleManifest" value="sampling-rules.json"/>
  </appSettings>
</configuration>
```

Exemple .NET Core appsettings.json — Règles d'échantillonnage

```
{
  "XRay": {
    "SamplingRuleManifest": "sampling-rules.json"
  }
}
```

Pour utiliser uniquement les règles locales, créez l'enregistreur avec une instruction `LocalizedSamplingStrategy`. Si vous avez des règles de sauvegarde configurées, supprimez cette configuration.

Exemple .NET global.asax — Règles d'échantillonnage locales

```
var recorder = new AWSXRayRecorderBuilder().WithSamplingStrategy(new
    LocalizedSamplingStrategy("samplingrules.json")).Build();
AWSXRayRecorder.InitializeInstance(recorder: recorder);
```

Exemple .NET Core Program.cs — Règles d'échantillonnage locales

```
var recorder = new AWSXRayRecorderBuilder().WithSamplingStrategy(new
    LocalizedSamplingStrategy("sampling-rules.json")).Build();
AWSXRayRecorder.InitializeInstance(configuration, recorder);
```

Journalisation (.NET)

Le SDK X-Ray pour .NET utilise le même mécanisme de journalisation que le [AWS SDK pour .NET](#). Si vous avez déjà configuré votre application pour enregistrer les AWS SDK pour .NET sorties, la même configuration s'applique aux sorties du SDK X-Ray pour .NET.

Pour configurer la journalisation, ajoutez une section de configuration nommée `aws` à votre fichier `App.config` ou `Web.config`.

Exemple Web.config - journalisation

```
...
<configuration>
  <configSections>
    <section name="aws" type="Amazon.AWSSection, AWSSDK.Core"/>
  </configSections>
  <aws>
    <logging logTo="Log4Net"/>
  </aws>
</configuration>
```

Pour plus d'informations, consultez [Configuration de votre application AWS SDK pour .NET](#) dans le Manuel du développeur AWS SDK pour .NET .

Journalisation (.NET Core)

Le kit de développement X-Ray pour .NET utilise les mêmes options de journalisation que le [AWS SDK pour .NET](#). Pour configurer la journalisation pour les applications .NET Core, transmettez l'option de journalisation à la `AWSXRayRecorder.RegisterLogger` méthode.

Par exemple, pour utiliser log4net, créez un fichier de configuration qui définit l'enregistreur d'événements, le format de sortie et l'emplacement du fichier.

Exemple .NET Core log4net.config

```
<?xml version="1.0" encoding="utf-8" ?>
<log4net>
  <appender name="FileAppender" type="log4net.Appender.FileAppender,log4net">
    <file value="c:\logs\sdk-log.txt" />
    <layout type="log4net.Layout.PatternLayout">
      <conversionPattern value="%date [%thread] %level %logger - %message%newline" />
    </layout>
  </appender>
  <logger name="Amazon">
    <level value="DEBUG" />
    <appender-ref ref="FileAppender" />
  </logger>
</log4net>
```

Ensuite, créez l'enregistreur d'événements et appliquez la configuration dans le code de programme.

Example .NET Core Program.cs — Journalisation

```
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)
    {
        ...
    }
}
```

Pour plus d'informations sur la configuration de log4net, consultez [Configuration](#) sur [logging.apache.org](#).

Variables d'environnement

Vous pouvez utiliser des variables d'environnement pour configurer le SDK X-Ray pour .NET. Le kit SDK prend en charge les variables suivantes.

- `AWS_XRAY_TRACING_NAME`— Définissez un nom de service que le SDK utilise pour les segments. Remplace le nom de service que vous définissez sur la [stratégie d'attribution de noms de segment](#) du filtre servlet.
- `AWS_XRAY_DAEMON_ADDRESS`— Définissez l'hôte et le port de l'écouteur du daemon X-Ray. Par défaut, le SDK utilise à la fois `127.0.0.1:2000` les données de trace (UDP) et l'échantillonnage (TCP). Utilisez cette variable si vous avez configuré le démon pour qu'il [écoute sur un port différent](#) ou s'il s'exécute sur un autre hôte.

Format

- Même port — *address:port*
- Différents ports — `tcp:address:port` `udp:address:port`

- `AWS_XRAY_CONTEXT_MISSING`— Réglé sur `RUNTIME_ERROR` pour générer des exceptions lorsque votre code instrumenté tente d'enregistrer des données alors qu'aucun segment n'est ouvert.

Valeurs valides

- `RUNTIME_ERROR`— Lance une exception d'exécution.
- `LOG_ERROR`— Enregistrez une erreur et continuez (par défaut).
- `IGNORE_ERROR`— Ignorez l'erreur et continuez.

Des erreurs liées à des segments ou sous-segments manquants peuvent se produire lorsque vous essayez d'utiliser un client instrumenté dans un code de démarrage qui s'exécute lorsqu'aucune demande n'est ouverte, ou dans un code qui génère un nouveau thread.

Instrumentation des requêtes HTTP entrantes avec le SDK X-Ray pour .NET

Vous pouvez utiliser le SDK X-Ray pour suivre les requêtes HTTP entrantes traitées par votre application sur une EC2 instance Amazon EC2 ou Amazon ECS. AWS Elastic Beanstalk

Utilisez un gestionnaire de messages pour instrumenter les demandes HTTP entrantes. Lorsque vous ajoutez le gestionnaire de messages X-Ray à votre application, le SDK X-Ray pour .NET crée un segment pour chaque requête échantillonnée. Ce segment comprend la durée, la méthode et l'état de la demande HTTP. L'instrumentation supplémentaire crée des sous-segments sur ce segment.

Note

Pour les AWS Lambda fonctions, Lambda crée un segment pour chaque requête échantillonnée. Pour plus d'informations, consultez [AWS Lambda et AWS X-Ray](#).

Chaque segment porte un nom qui identifie votre application dans la carte des services. Le segment peut être nommé de manière statique ou vous pouvez configurer le SDK pour qu'il soit nommé dynamiquement en fonction de l'en-tête de l'hôte dans la demande entrante. La dénomination dynamique vous permet de regrouper les traces en fonction du nom de domaine indiqué dans la demande et d'appliquer un nom par défaut si le nom ne correspond pas au modèle attendu (par exemple, si l'en-tête de l'hôte est falsifié).

Demandes transmises

Si un équilibreur de charge ou un autre intermédiaire transmet une demande à votre application, X-Ray prend l'adresse IP du client depuis l'`X-Forwarded-For`-tête de la demande plutôt que depuis l'adresse IP source du paquet IP. L'adresse IP du client enregistrée pour une demande transférée peut être falsifiée, elle ne doit donc pas être fiable.

Le gestionnaire de messages crée un segment pour chaque demande entrante avec un bloc `http` contenant les informations suivantes :

- Méthode HTTP : `GET`, `POST`, `PUT`, `DELETE`, etc.
- Adresse du client : adresse IP du client qui a envoyé la demande.
- Code de réponse : code de réponse HTTP pour la demande terminée.
- Moment : heure de début (date de réception de la demande) et heure de fin (date d'envoi de la réponse).
- Agent utilisateur — Le formulaire `user-agent` de la demande.
- Longueur du contenu : `content-length` extrait de la réponse.

Sections

- [Instrumentation des demandes entrantes \(.NET\)](#)
- [Instrumentation des demandes entrantes \(.NET Core\)](#)
- [Configuration d'une stratégie d'attribution de noms de segment](#)

Instrumentation des demandes entrantes (.NET)

Pour instrumenter des demandes servies par votre application, appelez `RegisterXRay` dans la méthode `Init` de votre fichier `global.asax`.

Exemple `global.asax` - Gestionnaire de messages

```
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");
    }
}
```

Instrumentation des demandes entrantes (.NET Core)

Pour instrumenter les requêtes traitées par votre application, appelez `UseXRay` Method avant tout autre intergiciel dans la `Configure` méthode de votre classe `Startup`. Idéalement, le middleware X-Ray devrait être le premier intergiciel à traiter la demande et le dernier intergiciel à traiter la réponse dans le pipeline.

Note

Pour .NET Core 2.0, si vous avez une `UseExceptionHandler` méthode dans l'application, assurez-vous d'appeler `UseXRay` après la `UseExceptionHandler` méthode pour vous assurer que les exceptions sont enregistrées.

Exemple 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
    ...
}
```

La méthode `UseXRay` peut également accepter un [objet de configuration](#) comme deuxième argument.

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

Configuration d'une stratégie d'attribution de noms de segment

AWS X-Ray utilise un nom de service pour identifier votre application et la distinguer des autres applications, bases de données, ressources externes APIs et AWS ressources utilisées par votre application. Lorsque le SDK X-Ray génère des segments pour les demandes entrantes, il enregistre le nom du service de votre application dans le [champ du nom](#) du segment.

Le SDK X-Ray peut nommer les segments d'après le nom d'hôte dans l'en-tête de la requête HTTP. Cependant, cet en-tête peut être falsifié, ce qui peut entraîner la création de nœuds inattendus dans votre carte de service. Pour éviter que le SDK ne nomme les segments de manière incorrecte en raison de demandes contenant des en-têtes d'hôte falsifiés, vous devez spécifier un nom par défaut pour les demandes entrantes.

Si votre application traite des demandes pour plusieurs domaines, vous pouvez configurer le SDK pour qu'il utilise une stratégie de dénomination dynamique afin de refléter cela dans les noms des segments. Une stratégie de dénomination dynamique permet au SDK d'utiliser le nom d'hôte pour les demandes qui correspondent à un modèle attendu et d'appliquer le nom par défaut aux demandes qui ne le sont pas.

Par exemple, il se peut qu'une seule application envoie des demandes à trois sous-domaines : `www.example.com`, `api.example.com`, et `static.example.com`. Vous pouvez utiliser une stratégie de dénomination dynamique avec le modèle `*.example.com` pour identifier les segments de chaque sous-domaine avec un nom différent, ce qui permet d'obtenir trois nœuds de service sur la carte des services. Si votre application reçoit des demandes dont le nom d'hôte ne correspond pas au modèle, vous verrez un quatrième nœud sur la carte des services avec un nom de remplacement que vous spécifiez.

Pour utiliser le même nom pour tous les segments de la demande, spécifiez le nom de votre application lorsque vous initialisez le gestionnaire de messages, comme indiqué dans la [section précédente](#). L'effet est le même que celui obtenu en créant une [FixedSegmentNamingStrategy](#) et en la transmettant à la méthode `RegisterXRay`.

```
AWSXRayAspNet.RegisterXRay(this, new FixedSegmentNamingStrategy("MyApp"));
```

Note

Vous pouvez remplacer le nom de service par défaut que vous définissez avec la `AWS_XRAY_TRACING_NAME` variable d'environnement [???](#).

Une stratégie d'attribution de noms dynamique définit un modèle auquel doivent correspondre les noms d'hôte et un nom par défaut à utiliser si le nom d'hôte de la demande HTTP ne correspond pas au modèle. Pour nommer dynamiquement des segments, créez une stratégie [DynamicSegmentNamingStrategy](#) et transmettez-la à la méthode `RegisterXRay`.

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

Suivi des appels du AWS SDK avec le SDK X-Ray pour .NET

Lorsque votre application effectue des appels Services AWS pour stocker des données, écrire dans une file d'attente ou envoyer des notifications, le SDK X-Ray pour .NET suit les appels en aval [dans](#) des sous-segments. Les ressources tracées Services AWS et auxquelles vous accédez au sein de ces services (par exemple, un compartiment Amazon S3 ou une file d'attente Amazon SQS) apparaissent sous forme de nœuds en aval sur la carte de trace de la console X-Ray.

Vous pouvez instrumenter tous vos AWS SDK pour .NET clients en les appelant `RegisterXRayForAllServices` avant de les créer.

Exemple `SampleController.cs` - Instrumentation du client DynamoDB

```
using Amazon;  
using Amazon.Util;  
using Amazon.DynamoDBv2;  
using Amazon.DynamoDBv2.DocumentModel;
```

```
using Amazon.XRay.Recorder.Core;
using Amazon.XRay.Recorder.Handlers.AwsSdk;

namespace SampleEBWebApplication.Controllers
{
    public class SampleController : ApiController
    {
        AWSSDKHandler.RegisterXRayForAllServices();
        private static readonly Lazy<AmazonDynamoDBClient> LazyDdbClient = new
        Lazy<AmazonDynamoDBClient>(() =>
        {
            var client = new AmazonDynamoDBClient(EC2InstanceMetadata.Region ??
            RegionEndpoint.USEast1);
            return client;
        });
    }
}
```

Pour instrumenter des clients pour certains services et pas d'autres, appelez `RegisterXRay` au lieu de `RegisterXRayForAllServices`. Remplacez le texte en surbrillance par le nom de l'interface du client du service.

```
AWSSDKHandler.RegisterXRay<IAmazonDynamoDB>()
```

Pour tous les services, vous pouvez voir le nom de l'API appelée dans la console X-Ray. Pour un sous-ensemble de services, le SDK X-Ray ajoute des informations au segment afin d'améliorer la granularité de la carte des services.

Par exemple, lorsque vous passez un appel avec un client DynamoDB instrumenté, le SDK ajoute le nom de la table au segment pour les appels qui ciblent une table. Dans la console, chaque table apparaît sous la forme d'un nœud distinct dans la carte des services, avec un nœud DynamoDB générique pour les appels qui ne ciblent pas une table.

Exemple Sous-segment pour un appel à DynamoDB pour enregistrer un élément

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

Lorsque vous accédez aux ressources nommées, les appels vers les services suivants créent des nœuds supplémentaires dans la cartographie des services. Les appels qui ne ciblent pas de ressources spécifiques créent un nœud générique pour le service.

- Amazon DynamoDB — Nom de la table
- Amazon Simple Storage Service : nom du compartiment et de la clé
- Amazon Simple Queue Service — Nom de la file d'attente

Suivi des appels vers les services Web HTTP en aval avec le SDK X-Ray pour .NET

Lorsque votre application appelle des microservices ou du protocole HTTP public APIs, vous pouvez utiliser la méthode `GetResponseTraced` du SDK X-Ray pour .NET pour `System.Net.HttpWebRequest` instrumenter ces appels et ajouter l'API au graphe de service en tant que service en aval.

Exemple `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();
}
```

Pour les appels asynchrones, utilisez `GetAsyncResponseTraced`.

```
request.GetAsyncResponseTraced();
```

Si vous utilisez [system.net.http.httpclient](#), utilisez le gestionnaire de délégation `HttpClientXRayTracingHandler` pour enregistrer les appels.

Exemple 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);
}
```

Lorsque vous instrumentez un appel à une API Web en aval, le SDK X-Ray pour .NET enregistre un sous-segment contenant des informations sur la requête et la réponse HTTP. X-Ray utilise le sous-segment pour générer un segment inféré pour l'API.

Exemple Sous-segment pour un appel HTTP en aval

```
{
  "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
    }
  }
}
```

```
}
```

Exemple Segment déduit pour un appel HTTP en aval

```
{
  "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
}
```

Suivi des requêtes SQL avec le SDK X-Ray pour .NET

Le SDK X-Ray pour .NET fournit une classe wrapper `System.Data.SqlClient.SqlCommand` pour, `TraceableSqlCommand` named, que vous pouvez utiliser à la place de. `SqlCommand` Vous pouvez initialiser une commande SQL avec la classe `TraceableSqlCommand`.

Suivi des requêtes SQL avec des méthodes synchrones et asynchrones

Les exemples suivants montrent comment utiliser `TraceableSqlCommand` pour suivre automatiquement les requêtes SQL Server de manière synchrone et asynchrone.

Exemple **Controller.cs** - Instrumentation de client SQL (synchrone)

```
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();
        }
}
```

Vous pouvez exécuter la requête de manière asynchrone à l'aide de la méthode `ExecuteReaderAsync`.

Exemple **Controller.cs** - Instrumentation de client SQL (asynchrone)

```
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();
        }
}
```

Collecte de requêtes SQL sur SQL Server

Vous pouvez activer la capture de `SqlCommand.CommandText` dans le cadre du sous-segment créé par votre requête SQL. `SqlCommand.CommandText` apparaît en tant que champ `sanitized_query` dans le sous-segment JSON. Par défaut, cette fonction est désactivée pour des raisons de sécurité.

Note

N'activez pas la fonction de collecte si vous incluez des informations sensibles sous forme de texte clair dans vos requêtes SQL.

Vous pouvez activer la collection de requêtes SQL de deux manières :

- Définissez la propriété `CollectSqlQueries` sur `true` dans la configuration globale de votre application.
- Définissez le paramètre `collectSqlQueries` dans l'instance `TraceableSqlCommand` sur `true` pour collecter les appels au sein de l'instance.

Activer la `CollectSqlQueries` propriété globale

Les exemples suivants montrent comment activer la propriété `CollectSqlQueries` pour .NET et .NET Core.

.NET

Pour définir la propriété `CollectSqlQueries` sur `true` dans la configuration globale de votre application dans .NET, modifiez la valeur de `appsettings` de votre fichier `Web.config` ou `App.config`, comme indiqué.

Exemple **App.config** Ou **Web.config** — Activez la collecte de requêtes SQL de manière globale

```
<configuration>
<appSettings>
  <add key="CollectSqlQueries" value="true">
</appSettings>
</configuration>
```

.NET Core

Pour définir la `CollectSqlQueries` propriété sur `true` dans la configuration globale de votre application dans .NET Core, modifiez votre `appsettings.json` fichier sous la touche `X-Ray`, comme indiqué.

Exemple **appsettings.json**— Active la collecte de requêtes SQL de manière globale

```
{
  "XRay": {
    "CollectSqlQueries": "true"
  }
}
```


Activez le collectSqlQueries paramètre

Vous pouvez définir le paramètre `collectSqlQueries` dans l'instance `TraceableSqlCommand` sur `true` afin de collecter le texte de la requête SQL pour les requêtes SQL Server effectuées à l'aide de cette instance. La définition du paramètre sur `false` désactive la fonction `CollectSqlQuery` pour l'instance `TraceableSqlCommand`.

Note

La valeur de `collectSqlQueries` dans l'instance `TraceableSqlCommand` remplace la valeur définie dans la configuration globale de la propriété `CollectSqlQueries`.

Exemple Exemple **Controller.cs** — Activer la collecte de requêtes SQL pour l'instance

```
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();
        }
    }
}
```

Création de sous-segments supplémentaires

Les sous-segments étendent le [segment](#) d'une trace avec des détails sur le travail effectué afin de répondre à une demande. Chaque fois que vous passez un appel avec un client instrumenté, le SDK X-Ray enregistre les informations générées dans un sous-segment. Vous pouvez créer des sous-segments supplémentaires pour regrouper d'autres sous-segments, pour mesurer les performances d'une section de code ou pour enregistrer des annotations et des métadonnées.

Pour gérer des sous-segments, utilisez les méthodes `BeginSubsegment` et `EndSubsegment`. Exécutez des travaux dans le sous-segment dans un bloc `try` et utilisez `AddException` pour

suivre les exceptions. Appelez `EndSubsegment` dans un bloc `finally` pour veiller à ce que le segment soit fermé.

Exemple Controller.cs — Sous-segment personnalisé

```
AWSXRayRecorder.Instance.BeginSubsegment("custom method");
try
{
    DoWork();
}
catch (Exception e)
{
    AWSXRayRecorder.Instance.AddException(e);
}
finally
{
    AWSXRayRecorder.Instance.EndSubsegment();
}
```

Lorsque vous créez un sous-segment au sein d'un segment ou d'un autre sous-segment, le SDK X-Ray pour .NET génère un identifiant et enregistre l'heure de début et de fin.

Exemple Sous-segment avec des métadonnées

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

Ajoutez des annotations et des métadonnées aux segments avec le SDK X-Ray pour .NET

Vous pouvez enregistrer des informations supplémentaires sur les demandes, l'environnement ou votre application à l'aide d'annotations et de métadonnées. Vous pouvez ajouter des annotations et

des métadonnées aux segments créés par le SDK X-Ray ou aux sous-segments personnalisés que vous créez.

Les annotations sont des paires clé-valeur avec des chaînes, des nombres ou des valeurs booléennes. Les annotations sont indexées pour être utilisées avec les expressions de [filtre](#). Utilisez les annotations pour enregistrer les données que vous souhaitez utiliser pour regrouper les suivis dans la console ou lors de l'appel de l'API [GetTraceSummaries](#).

Les métadonnées sont des paires clé-valeur qui peuvent contenir des valeurs de n'importe quel type, y compris des objets et des listes, mais qui ne sont pas indexées pour être utilisées avec des expressions de filtre. Utilisez les métadonnées pour enregistrer des données supplémentaires que vous souhaitez stocker dans le traçage, mais que vous n'avez pas besoin d'utiliser pour la recherche.

Sections

- [Enregistrement d'annotations avec le SDK X-Ray pour .NET](#)
- [Enregistrement de métadonnées avec le SDK X-Ray pour .NET](#)

Enregistrement d'annotations avec le SDK X-Ray pour .NET

Utilisez les annotations pour enregistrer les informations sur les segments ou sous-segments qui doivent être indexés pour la recherche.

Les éléments suivants sont requis pour toutes les annotations dans X-Ray :

Exigences liées aux annotations

- Clés — La clé d'une annotation X-Ray peut comporter jusqu'à 500 caractères alphanumériques. Vous ne pouvez pas utiliser d'espaces ou de symboles autres qu'un point ou un point (.)
- Valeurs — La valeur d'une annotation X-Ray peut comporter jusqu'à 1 000 caractères Unicode.
- Nombre d'annotations : vous pouvez utiliser jusqu'à 50 annotations par trace.

Pour enregistrer des annotations en dehors d'une fonction AWS Lambda

1. Obtenez une instance de `AWSXRayRecorder`.

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

2. Appelez `addAnnotation` avec une clé de type chaîne et une valeur de type booléen, `Int32`, `Int64`, `Double` ou chaîne.

```
recorder.AddAnnotation("mykey", "my value");
```

L'exemple suivant montre comment appeler `putAnnotation` avec une clé `String` qui inclut un point et une valeur booléenne, numérique ou chaîne.

```
document.putAnnotation("testkey.test", "my value");
```

Pour enregistrer des annotations dans une fonction AWS Lambda

Les segments et sous-segments d'une fonction Lambda sont gérés par l'environnement d'exécution Lambda. Si vous souhaitez ajouter une annotation à un segment ou à un sous-segment dans une fonction Lambda, vous devez effectuer les opérations suivantes :

1. Créez le segment ou le sous-segment dans la fonction Lambda.
2. Ajoutez l'annotation au segment ou au sous-segment.
3. Terminez le segment ou le sous-segment.

L'exemple de code suivant montre comment ajouter une annotation à un sous-segment dans une fonction Lambda :

```
#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();
}
```

```
}
```

Le SDK X-Ray enregistre les annotations sous forme de paires clé-valeur dans un annotations objet du document segmenté. Le fait d'appeler l'addAnnotationopération deux fois avec la même clé remplace une valeur précédemment enregistrée sur le même segment ou sous-segment.

Pour rechercher les suivis ayant des annotations avec des valeurs spécifiques, utilisez le mot clé `annotation[key]` dans une [expression de filtre](#).

Enregistrement de métadonnées avec le SDK X-Ray pour .NET

Utilisez les métadonnées pour enregistrer des informations sur des segments ou des sous-segments que vous n'avez pas besoin d'indexer pour les utiliser dans le cadre d'une recherche. Les valeurs de métadonnées peuvent être des chaînes, des nombres, des booléens ou tout autre objet pouvant être sérialisé dans un objet ou un tableau JSON.

Pour enregistrer les métadonnées

1. Obtenez une instance de `AWSXRayRecorder`, comme indiqué dans l'exemple de code suivant :

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

2. Appelez `AddMetadata` avec un espace de noms de chaîne, une clé de chaîne et une valeur d'objet, comme indiqué dans l'exemple de code suivant :

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

Vous pouvez également appeler l'AddMetadataopération en utilisant simplement une paire clé/valeur, comme illustré dans l'exemple de code suivant :

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

Si vous ne spécifiez aucune valeur pour l'espace de noms, le SDK X-Ray l'utilise. default Le fait d'appeler l'AddMetadataopération deux fois avec la même clé remplace une valeur précédemment enregistrée sur le même segment ou sous-segment.

Utilisation de Ruby

Il existe deux manières d'instrumenter votre application Ruby pour envoyer des traces à X-Ray :

- [AWS Distro for OpenTelemetry Ruby](#) : AWS distribution qui fournit un ensemble de bibliothèques open source permettant d'envoyer des métriques et des traces corrélées à plusieurs solutions de AWS surveillance, notamment Amazon CloudWatch et Amazon OpenSearch Service AWS X-Ray, via [AWS Distro](#) for Collector. OpenTelemetry
- [AWS X-Ray SDK pour](#) Ruby : ensemble de bibliothèques permettant de générer et d'envoyer des traces à X-Ray via le daemon [X-Ray](#).

Pour de plus amples informations, veuillez consulter [Choisir entre The AWS Distro for OpenTelemetry et X-Ray SDKs](#).

AWS Distro pour Ruby OpenTelemetry

Avec le AWS Distro for OpenTelemetry (ADOT) Ruby, vous pouvez instrumenter vos applications une seule fois et envoyer des métriques et des traces corrélées à plusieurs solutions de AWS surveillance CloudWatch, notamment Amazon AWS X-Ray et Amazon Service. OpenSearch L'utilisation de X-Ray avec ADOT nécessite deux composants : un OpenTelemetry SDK activé pour une utilisation avec X-Ray, et le AWS Distro for OpenTelemetry Collector activé pour une utilisation avec X-Ray.

Pour commencer, consultez la [documentation de AWS Distro for OpenTelemetry Ruby](#).

Pour plus d'informations sur l'utilisation de la AWS distribution pour OpenTelemetry with AWS X-Ray et d'autres Services AWS, consultez la section [AWS Distro for OpenTelemetry](#) ou la distribution pour la [AWS documentation](#). OpenTelemetry

Pour plus d'informations sur la prise en charge et l'utilisation des langues, consultez [AWS Observability on GitHub](#).

AWS X-Ray SDK pour Ruby

Le SDK X-Ray est une bibliothèque pour les applications Web Ruby qui fournit des classes et des méthodes permettant de générer et d'envoyer des données de trace au daemon X-Ray. Les données de suivi incluent des informations sur les requêtes HTTP entrantes traitées par l'application et les

appels que l'application envoie aux services en aval à l'aide du AWS SDK, de clients HTTP ou d'un client d'enregistrement actif. Vous pouvez également créer des segments manuellement et ajouter des informations de débogage dans les annotations et les métadonnées.

Vous pouvez télécharger le kit SDK en l'ajoutant à votre gemfile et en exécutant `bundle install`.

Exemple Gemfile

```
gem 'aws-sdk'
```

Si vous utilisez Rails, commencez par [ajouter le middleware du SDK X-Ray](#) pour suivre les demandes entrantes. Un filtre de demande crée un [segment](#). Lorsque le segment est ouvert, vous pouvez utiliser les méthodes du client du SDK pour ajouter des informations au segment et créer des sous-segments afin de suivre les appels en aval. Le kit de développement logiciel (SDK) enregistre aussi automatiquement les exceptions que votre application renvoie quand le segment est ouvert. Pour les applications non-Rails, vous pouvez [créer des segments manuellement](#).

Utilisez ensuite le SDK X-Ray pour instrumenter vos AWS SDK pour Ruby clients HTTP et SQL en [configurant l'enregistreur pour qu'il applique](#) des correctifs aux bibliothèques associées. Chaque fois que vous appelez une ressource Service AWS ou une ressource en aval avec un client instrumenté, le SDK enregistre les informations relatives à l'appel dans un sous-segment. Services AWS et les ressources auxquelles vous accédez au sein des services apparaissent sous forme de nœuds en aval sur la carte de trace pour vous aider à identifier les erreurs et les problèmes de limitation sur les connexions individuelles.

Une fois que vous êtes familiarisé avec le SDK, personnalisez son comportement en [configurant l'enregistreur](#). Vous pouvez ajouter des plug-ins pour enregistrer les données sur les ressources de calcul exécutant votre application, personnaliser le comportement d'échantillonnage en définissant des règles d'échantillonnage et fournir un enregistreur d'événements afin d'afficher plus ou moins d'informations du kit SDK dans vos journaux d'application.

Enregistrez les informations supplémentaires sur les demandes et le travail que votre application effectue dans les [annotations et les métadonnées](#). Les annotations sont de simples paires clé-valeur, indexées en vue de leur utilisation avec les [expressions de filtre](#), de telle sorte que vous pouvez explorer les suivis qui contiennent des données spécifiques. Les entrées des métadonnées sont moins restrictives et peuvent enregistrer des objets et tableaux entiers (tout ce qui peut être sérialisé en JSON).

Annotations et métadonnées

Les annotations et les métadonnées sont du texte arbitraire que vous ajoutez aux segments avec le SDK X-Ray. Les annotations sont indexées pour être utilisées avec les expressions de filtre. Les métadonnées ne sont pas indexées, mais peuvent être consultées dans le segment brut à l'aide de la console ou de l'API X-Ray. Toute personne à qui vous accordez un accès en lecture à X-Ray peut consulter ces données.

Lorsque vous avez un grand nombre de clients instrumentés dans votre code, un seul segment de demande peut contenir un grand nombre de sous-segments, un par appel effectué à l'aide d'un client instrumenté. Vous pouvez organiser et grouper les sous-segments en enveloppant les appels clients dans des [sous-segments personnalisés](#). Vous pouvez créer un sous-segment personnalisé pour une fonction complète ou une quelconque section du code, puis enregistrer les métadonnées et les annotations sur le sous-segment au lieu de tout écrire sur le segment parent.

Pour consulter la documentation de référence sur les classes et les méthodes du SDK, consultez le manuel de référence de l'[AWS X-Ray API SDK for Ruby](#).

Prérequis

Le SDK X-Ray nécessite Ruby 2.3 ou version ultérieure et est compatible avec les bibliothèques suivantes :

- AWS SDK pour Ruby version 3.0 ou ultérieure
- Rails version 5.1 ou version ultérieure

Configuration du SDK X-Ray pour Ruby

Le SDK X-Ray pour Ruby possède une classe `XRayRecorder` nommée qui fournit l'enregistreur global. Vous pouvez configurer l'enregistreur mondial afin qu'il personnalise l'intergiciel qui crée des segments pour les appels HTTP entrants.

Sections

- [Plug-ins de service](#)
- [Règles d'échantillonnage](#)
- [Journalisation](#)

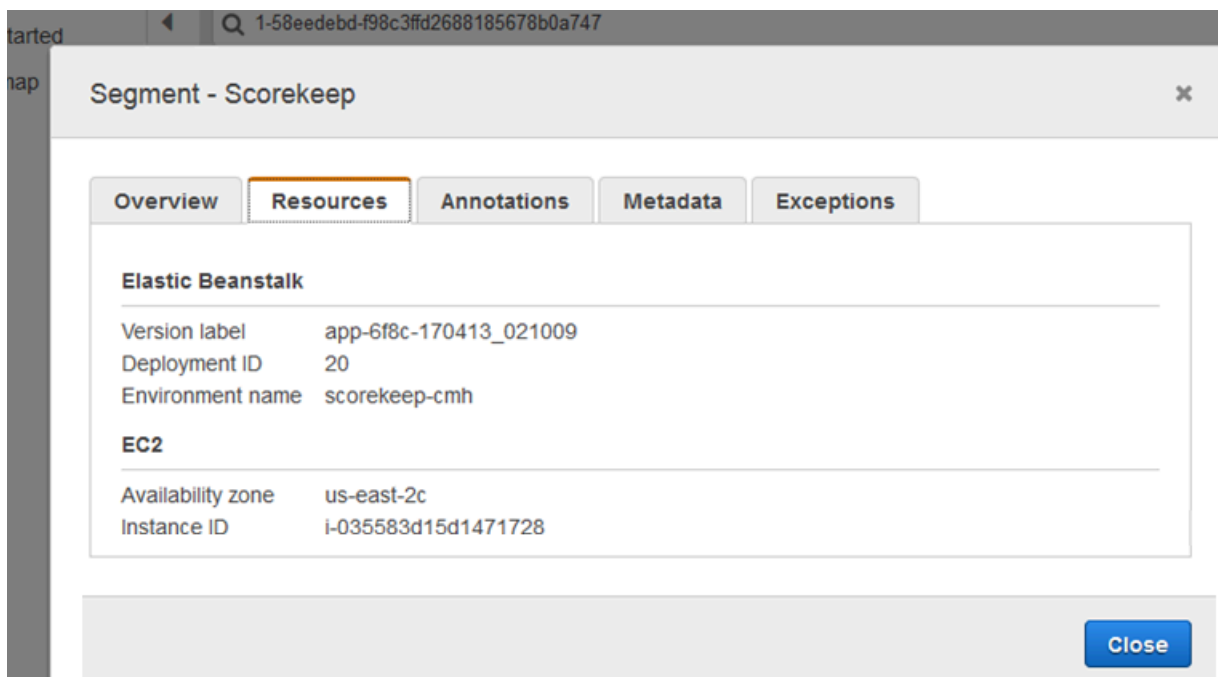
- [Configuration de l'enregistreur dans le code](#)
- [Configuration de l'enregistreur avec Rails](#)
- [Variables d'environnement](#)

Plug-ins de service

Permet plugins d'enregistrer des informations sur le service hébergeant votre application.

Plugins

- Amazon EC2 — `ec2` ajoute l'ID de l'instance et la zone de disponibilité.
- Elastic `elastic_beanstalk` Beanstalk : ajoute le nom de l'environnement, l'étiquette de version et l'ID de déploiement.
- Amazon ECS — `ecs` ajoute l'ID du conteneur.



Pour utiliser des plug-ins, spécifiez-les dans l'objet de configuration que vous transmettez à l'enregistreur.

Exemple `main.rb` — Configuration du plugin

```
my_plugins = %I[ec2 elastic_beanstalk]
```

```
config = {  
    plugins: my_plugins,  
    name: 'my app',  
}  
  
XRay.recorder.configure(config)
```

Vous pouvez également utiliser des [variables d'environnement](#), qui ont la priorité sur les valeurs définies dans le code, pour configurer l'enregistreur.

Le SDK utilise également les paramètres du plugin pour définir le `origin` champ du segment. Cela indique le type de AWS ressource qui exécute votre application. Lorsque vous utilisez plusieurs plugins, le SDK utilise l'ordre de résolution suivant pour déterminer l'origine : ElasticBeanstalk > EKS > ECS > EC2.

Règles d'échantillonnage

Le SDK utilise les règles d'échantillonnage que vous définissez dans la console X-Ray pour déterminer les demandes à enregistrer. La règle par défaut suit la première demande chaque seconde, et 5 % de toutes les demandes supplémentaires provenant de tous les services envoient des traces à X-Ray. [Créez des règles supplémentaires dans la console X-Ray](#) pour personnaliser la quantité de données enregistrées pour chacune de vos applications.

Le SDK applique les règles personnalisées dans l'ordre dans lequel elles sont définies. Si une demande correspond à plusieurs règles personnalisées, le SDK applique uniquement la première règle.

Note

Si le SDK ne parvient pas à accéder à X-Ray pour obtenir des règles d'échantillonnage, il revient à une règle locale par défaut concernant la première demande chaque seconde, et 5 % de toutes les demandes supplémentaires par hôte. Cela peut se produire si l'hôte n'est pas autorisé à appeler `sampling APIs` ou ne peut pas se connecter au daemon X-Ray, qui agit comme un proxy TCP pour les appels d'API effectués par le SDK.

Vous pouvez également configurer le SDK pour charger des règles d'échantillonnage à partir d'un document JSON. Le SDK peut utiliser les règles locales comme solution de rechange dans les cas où l'échantillonnage X-Ray n'est pas disponible, ou utiliser exclusivement les règles locales.

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
  }
}
```

Cet exemple définit une règle personnalisée et une règle par défaut. La règle personnalisée applique un taux d'échantillonnage de 5 % sans nombre minimum de demandes à suivre pour les chemins sous-jacents. `/api/move/` La règle par défaut suit la première demande chaque seconde et 10 % des demandes supplémentaires.

L'inconvénient de définir des règles localement est que la cible fixe est appliquée par chaque instance de l'enregistreur indépendamment, au lieu d'être gérée par le service X-Ray. Au fur et à mesure que vous déployez de nouveaux hôtes, le taux fixe est multiplié, ce qui complique le contrôle de la quantité de données enregistrées.

Pour configurer les règles de sauvegarde, définissez un hachage pour le document dans l'objet de configuration que vous transmettez à l'enregistreur.

Example main.rb — Configuration des règles de sauvegarde

```
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)
```

Pour stocker les règles d'échantillonnage de manière indépendante, définissez le hachage dans un fichier distinct et demandez au fichier de l'extraire dans votre application.

Exemple config/sampling-rules.rb

```
my_sampling_rules = {  
  version: 1,  
  default: {  
    fixed_target: 1,  
    rate: 0.1  
  }  
}
```

Exemple main.rb — Règle d'échantillonnage à partir d'un fichier

```
require 'aws-xray-sdk'  
require 'config/sampling-rules.rb'  
  
config = {  
  sampling_rules: my_sampling_rules,  
  name: 'my app',  
}  
XRay.recorder.configure(config)
```

Pour utiliser uniquement des règles locales, exigez les règles d'échantillonnage et configurez le `LocalSampler`.

Exemple main.rb — Échantillonnage de règles locales

```
require 'aws-xray-sdk'  
require 'aws-xray-sdk/sampling/local/sampler'  
  
config = {  
  sampler: LocalSampler.new,
```

```
name: 'my app',  
}  
XRay.recorder.configure(config)
```

Vous pouvez également configurer l'enregistreur mondial afin qu'il désactive l'échantillonnage et instrumente toutes les demandes entrantes.

Exemple main.rb — Désactive l'échantillonnage

```
require 'aws-xray-sdk'  
config = {  
  sampling: false,  
  name: 'my app',  
}  
XRay.recorder.configure(config)
```

Journalisation

Par défaut, l'enregistreur place les événements de niveau informations dans `$stdout`. Vous pouvez personnaliser la journalisation en définissant un [enregistreur d'événements](#) dans l'objet de configuration que vous transmettez à l'enregistreur.

Exemple main.rb — Journalisation

```
require 'aws-xray-sdk'  
config = {  
  logger: my_logger,  
  name: 'my app',  
}  
XRay.recorder.configure(config)
```

Utilisez les journaux de débogage pour identifier les problèmes, tels que des sous-segments ouverts, lorsque vous [générez manuellement des sous-segments](#).

Configuration de l'enregistreur dans le code

D'autres paramètres sont disponibles à partir de la méthode `configure` dans `XRay.recorder`.

- `context_missing`— Régulé sur `LOG_ERROR` pour éviter de générer des exceptions lorsque votre code instrumenté tente d'enregistrer des données alors qu'aucun segment n'est ouvert.

- `daemon_address`— Définissez l'hôte et le port de l'écouteur du daemon X-Ray.
- `name`— Définissez un nom de service que le SDK utilise pour les segments.
- `naming_pattern`— Définissez un modèle de nom de domaine pour utiliser la [dénomination dynamique](#).
- `plugins`— Enregistrez des informations sur les AWS ressources de votre application à l'aide de [plugins](#).
- `sampling`— Réglez sur `false` pour désactiver l'échantillonnage.
- `sampling_rules`— Définissez le hachage contenant vos [règles d'échantillonnage](#).

Exemple `main.rb` — Désactive les exceptions manquantes dans le contexte

```
require 'aws-xray-sdk'
config = {
  context_missing: 'LOG_ERROR'
}

XRay.recorder.configure(config)
```

Configuration de l'enregistreur avec Rails

Si vous utilisez l'infrastructure Rails, vous pouvez configurer les options sur l'enregistreur mondial dans un fichier Ruby sous `app_root/initializers`. Le SDK X-Ray prend en charge une clé de configuration supplémentaire à utiliser avec Rails.

- `active_record`— Paramétré sur `true` pour enregistrer les sous-segments des transactions de base de données Active Record.

Configurez les paramètres disponibles dans un objet de configuration nommé `Rails.application.config.xray`.

Exemple `config/initializers/aws_xray.rb`

```
Rails.application.config.xray = {
  name: 'my app',
  patch: %I[net_http aws_sdk],
  active_record: true
}
```

Variables d'environnement

Vous pouvez utiliser des variables d'environnement pour configurer le SDK X-Ray pour Ruby. Le kit SDK prend en charge les variables suivantes:

- `AWS_XRAY_TRACING_NAME`— Définissez un nom de service que le SDK utilise pour les segments. Remplace le nom de service que vous définissez sur la [stratégie d'attribution de noms de segment](#) du filtre servlet.
- `AWS_XRAY_DAEMON_ADDRESS`— Définissez l'hôte et le port de l'écouteur du daemon X-Ray. Par défaut, le SDK envoie les données de suivi à `127.0.0.1:2000`. Utilisez cette variable si vous avez configuré le démon pour qu'il [écoute sur un port différent](#) ou s'il s'exécute sur un autre hôte.
- `AWS_XRAY_CONTEXT_MISSING`— Réglé sur `RUNTIME_ERROR` pour générer des exceptions lorsque votre code instrumenté tente d'enregistrer des données alors qu'aucun segment n'est ouvert.

Valeurs valides

- `RUNTIME_ERROR`— Lance une exception d'exécution.
- `LOG_ERROR`— Enregistrez une erreur et continuez (par défaut).
- `IGNORE_ERROR`— Ignorez l'erreur et continuez.

Des erreurs liées à des segments ou sous-segments manquants peuvent se produire lorsque vous essayez d'utiliser un client instrumenté dans un code de démarrage qui s'exécute lorsqu'aucune demande n'est ouverte, ou dans un code qui génère un nouveau thread.

Les variables d'environnement remplacent les valeurs définies dans le code.

Suivi des demandes entrantes avec le kit de développement X-Ray SDK for Ruby Middleware

Vous pouvez utiliser le SDK X-Ray pour suivre les requêtes HTTP entrantes traitées par votre application sur une EC2 instance Amazon EC2 ou Amazon ECS. AWS Elastic Beanstalk

Si vous utilisez Rails, servez-vous de l'intergiciel Rails pour instrumenter les demandes HTTP entrantes. Lorsque vous ajoutez le middleware à votre application et que vous configurez un nom de segment, le SDK X-Ray pour Ruby crée un segment pour chaque requête échantillonnée. Tous les segments créés par une instrumentation supplémentaire deviennent des sous-segments du

segment au niveau de la demande qui fournit des informations sur la demande et la réponse HTTP. Ces informations comprennent la durée, la méthode et l'état de la demande.

Chaque segment porte un nom qui identifie votre application dans la carte des services. Le segment peut être nommé de manière statique ou vous pouvez configurer le SDK pour qu'il soit nommé dynamiquement en fonction de l'en-tête de l'hôte dans la demande entrante. La dénomination dynamique vous permet de regrouper les traces en fonction du nom de domaine indiqué dans la demande et d'appliquer un nom par défaut si le nom ne correspond pas au modèle attendu (par exemple, si l'en-tête de l'hôte est falsifié).

Demandes transmises

Si un équilibreur de charge ou un autre intermédiaire transmet une demande à votre application, X-Ray prend l'adresse IP du client depuis l'`X-Forwarded-For`-tête de la demande plutôt que depuis l'adresse IP source du paquet IP. L'adresse IP du client enregistrée pour une demande transférée peut être falsifiée, elle ne doit donc pas être fiable.

Lorsqu'une demande est transmise, le SDK définit un champ supplémentaire dans le segment pour l'indiquer. Si le segment contient le champ `x_forwarded_for` défini sur `true`, l'adresse IP du client a été extraite de l'`X-Forwarded-For`-tête de la requête HTTP.

L'intergiciel crée un segment pour chaque demande entrante avec un bloc `http` contenant les informations suivantes :

- Méthode HTTP : GET, POST, PUT, DELETE, etc.
- Adresse du client : adresse IP du client qui a envoyé la demande.
- Code de réponse : code de réponse HTTP pour la demande terminée.
- Moment : heure de début (date de réception de la demande) et heure de fin (date d'envoi de la réponse).
- Agent utilisateur — Le formulaire `user-agent` de la demande.
- Longueur du contenu : `content-length` extrait de la réponse.

Utilisation de l'intergiciel Rails

Pour utiliser l'intergiciel, mettez à jour votre `gemfile` pour inclure le [railtie](#) requis.

Exemple Gemfile - Rails

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

Pour utiliser le middleware, vous devez également [configurer l'enregistreur](#) avec un nom qui représente l'application dans la carte de trace.

Exemple config/initializers/aws_xray.rb

```
Rails.application.config.xray = {  
  name: 'my app'  
}
```

Instrumentation manuelle du code

Si vous n'utilisez pas Rails, créez des segments manuellement. Vous pouvez créer un segment pour chaque demande entrante, ou créer des segments autour de clients HTTP ou AWS SDK corrigés afin de fournir un contexte permettant à l'enregistreur d'ajouter des sous-segments.

```
# 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
```

Configuration d'une stratégie d'attribution de noms de segment

AWS X-Ray utilise un nom de service pour identifier votre application et la distinguer des autres applications, bases de données, ressources externes APIs et AWS ressources utilisées par votre application. Lorsque le SDK X-Ray génère des segments pour les demandes entrantes, il enregistre le nom du service de votre application dans le [champ du nom](#) du segment.

Le SDK X-Ray peut nommer les segments d'après le nom d'hôte dans l'en-tête de la requête HTTP. Cependant, cet en-tête peut être falsifié, ce qui peut entraîner la création de nœuds inattendus dans votre carte de service. Pour éviter que le SDK ne nomme les segments de manière incorrecte en raison de demandes contenant des en-têtes d'hôte falsifiés, vous devez spécifier un nom par défaut pour les demandes entrantes.

Si votre application traite des demandes pour plusieurs domaines, vous pouvez configurer le SDK pour qu'il utilise une stratégie de dénomination dynamique afin de refléter cela dans les noms des segments. Une stratégie de dénomination dynamique permet au SDK d'utiliser le nom d'hôte pour les demandes qui correspondent à un modèle attendu et d'appliquer le nom par défaut aux demandes qui ne le sont pas.

Par exemple, il se peut qu'une seule application envoie des demandes à trois sous-domaines : `www.example.com`, `api.example.com`, et `static.example.com`. Vous pouvez utiliser une stratégie de dénomination dynamique avec le modèle `*.example.com` pour identifier les segments de chaque sous-domaine avec un nom différent, ce qui permet d'obtenir trois nœuds de service sur la carte des services. Si votre application reçoit des demandes dont le nom d'hôte ne correspond pas au modèle, vous verrez un quatrième nœud sur la carte des services avec un nom de remplacement que vous spécifiez.

Pour utiliser le même nom pour tous les segments de demande, spécifiez le nom de votre application lorsque vous configurez l'enregistreur, comme illustré dans les [sections précédentes](#).

Une stratégie d'attribution de noms dynamique définit un modèle auquel doivent correspondre les noms d'hôte et un nom par défaut à utiliser si le nom d'hôte de la demande HTTP ne correspond pas au modèle. Pour nommer des segments de façon dynamique, spécifiez un modèle d'affectation de noms dans le hachage de configuration.

Exemple `main.rb` — Dénomination dynamique

```
config = {
```

```
naming_pattern: '*mydomain*',  
name: 'my app',  
}  
  
XRay.recorder.configure(config)
```

Vous pouvez utiliser '*' dans le modèle pour correspondre à n'importe quelle chaîne ou '?' pour correspondre à un seul caractère.

Note

Vous pouvez remplacer le nom de service par défaut que vous définissez avec la `AWS_XRAY_TRACING_NAME` variable d'environnement [???](#).

Correction des bibliothèques pour instrumenter les appels en aval

Pour instrumenter les appels en aval, utilisez le SDK X-Ray pour Ruby afin de patcher les bibliothèques utilisées par votre application. Le SDK X-Ray pour Ruby peut appliquer des correctifs aux bibliothèques suivantes.

Bibliothèques prises en charge

- [net/http](#)— Clients HTTP de l'instrument.
- [aws-sdk](#)— AWS SDK pour Ruby Clients de l'instrument.

Lorsque vous utilisez une bibliothèque patchée, le SDK X-Ray pour Ruby crée un sous-segment pour l'appel et enregistre les informations issues de la demande et de la réponse. Un segment doit être disponible pour que le kit SDK puisse créer le sous-segment à partir de l'intergiciel SDK ou d'un appel à `XRay.recorder.begin_segment`.

Pour patcher des bibliothèques, spécifiez-les dans l'objet de configuration que vous transmettez à l'enregistreur X-Ray.

Exemple main.rb — Bibliothèques de correctifs

```
require 'aws-xray-sdk'
```

```
config = {
  name: 'my app',
  patch: %I[net_http aws_sdk]
}

XRay.recorder.configure(config)
```

Suivi des appels du AWS SDK avec le SDK X-Ray pour Ruby

Lorsque votre application fait des appels Services AWS pour stocker des données, écrire dans une file d'attente ou envoyer des notifications, le SDK X-Ray pour Ruby suit les appels en aval [dans](#) des sous-segments. Les ressources tracées Services AWS et auxquelles vous accédez au sein de ces services (par exemple, un compartiment Amazon S3 ou une file d'attente Amazon SQS) apparaissent sous forme de nœuds en aval sur la carte de suivi de la console X-Ray.

Le SDK X-Ray pour Ruby instrumente automatiquement AWS tous les clients du SDK lorsque [vous appliquez un correctif aws-sdk](#) à la bibliothèque. Vous ne pouvez pas instrumenter des clients individuels.

Pour tous les services, vous pouvez voir le nom de l'API appelée dans la console X-Ray. Pour un sous-ensemble de services, le SDK X-Ray ajoute des informations au segment afin d'améliorer la granularité de la carte des services.

Par exemple, lorsque vous passez un appel avec un client DynamoDB instrumenté, le SDK ajoute le nom de la table au segment pour les appels qui ciblent une table. Dans la console, chaque table apparaît sous la forme d'un nœud distinct dans la carte des services, avec un nœud DynamoDB générique pour les appels qui ne ciblent pas une table.

Exemple Sous-segment pour un appel à DynamoDB pour enregistrer un élément

```
{
  "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",
  }
}

```

Lorsque vous accédez aux ressources nommées, les appels vers les services suivants créent des nœuds supplémentaires dans la cartographie des services. Les appels qui ne ciblent pas de ressources spécifiques créent un nœud générique pour le service.

- Amazon DynamoDB — Nom de la table
- Amazon Simple Storage Service : nom du compartiment et de la clé
- Amazon Simple Queue Service — Nom de la file d'attente

Génération de sous-segments personnalisés avec le SDK X-Ray

Les sous-segments étendent le [segment](#) d'une trace avec des détails sur le travail effectué afin de répondre à une demande. Chaque fois que vous passez un appel avec un client instrumenté, le SDK X-Ray enregistre les informations générées dans un sous-segment. Vous pouvez créer des sous-segments supplémentaires pour regrouper d'autres sous-segments, pour mesurer les performances d'une section de code ou pour enregistrer des annotations et des métadonnées.

Pour gérer des sous-segments, utilisez les méthodes `begin_subsegment` et `end_subsegment`.

```

subsegment = XRay.recorder.begin_subsegment name: 'annotations', namespace: 'remote'
my_annotations = { id: 12345 }
subsegment.annotations.update my_annotations
XRay.recorder.end_subsegment

```

Pour créer un sous-segment pour une fonction, encapsulez-le dans un appel à `XRay.recorder.capture`.

```

XRay.recorder.capture('name_for_subsegment') do |subsegment|
  resp = myfunc() # myfunc is your function
  subsegment.annotations.update k1: 'v1'
  resp
end

```

```
end
```

Lorsque vous créez un sous-segment au sein d'un segment ou d'un autre sous-segment, le SDK X-Ray génère un identifiant pour celui-ci et enregistre l'heure de début et l'heure de fin.

Exemple Sous-segment avec des métadonnées

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

Ajoutez des annotations et des métadonnées aux segments avec le SDK X-Ray pour Ruby

Vous pouvez enregistrer des informations supplémentaires sur les demandes, l'environnement ou votre application à l'aide d'annotations et de métadonnées. Vous pouvez ajouter des annotations et des métadonnées aux segments créés par le SDK X-Ray ou aux sous-segments personnalisés que vous créez.

Les annotations sont des paires clé-valeur avec des chaînes, des nombres ou des valeurs booléennes. Les annotations sont indexées pour être utilisées avec les expressions de [filtre](#). Utilisez les annotations pour enregistrer les données que vous souhaitez utiliser pour regrouper les suivis dans la console ou lors de l'appel de l'API [GetTraceSummaries](#).

Les métadonnées sont des paires clé-valeur qui peuvent contenir des valeurs de n'importe quel type, y compris des objets et des listes, mais qui ne sont pas indexées pour être utilisées avec des expressions de filtre. Utilisez les métadonnées pour enregistrer des données supplémentaires que vous souhaitez stocker dans le traçage, mais dont vous n'avez pas besoin pour effectuer une recherche.

En plus des annotations et des métadonnées, vous pouvez également [enregistrer les chaînes d'ID utilisateur](#) sur des segments. Les utilisateurs sont enregistrés dans un champ distinct sur les segments et sont indexés pour être utilisés dans le cadre de la recherche.

Sections

- [Enregistrer des annotations avec le SDK X-Ray pour Ruby](#)
- [Enregistrement de métadonnées avec le SDK X-Ray pour Ruby](#)
- [Utilisateur enregistrant IDs avec le SDK X-Ray pour Ruby](#)

Enregistrer des annotations avec le SDK X-Ray pour Ruby

Utilisez les annotations pour enregistrer les informations sur les segments ou sous-segments qui doivent être indexés pour la recherche.

Exigences liées aux annotations

- Clés — La clé d'une annotation X-Ray peut comporter jusqu'à 500 caractères alphanumériques. Vous ne pouvez pas utiliser d'espaces ou de symboles autres qu'un point ou un point (.)
- Valeurs — La valeur d'une annotation X-Ray peut comporter jusqu'à 1 000 caractères Unicode.
- Nombre d'annotations : vous pouvez utiliser jusqu'à 50 annotations par trace.

Pour enregistrer les annotations

1. Obtenez une référence au segment ou sous-segment en cours à partir d'`xray_recorder`.

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

or

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

2. Appelez `update` avec une valeur de hachage.

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

L'exemple suivant montre comment appeler `update` avec une clé d'annotation contenant un point.

```
my_annotations = { testkey.test: 12345 }  
document.annotations.update my_annotations
```

Le kit de développement logiciel enregistre les annotations sous forme de paires clé-valeur dans un objet `annotations` du document de segment. Si vous appelez deux fois `add_annotations` avec la même clé, les valeurs précédemment enregistrées sur le même segment ou sous-segment sont remplacées.

Pour rechercher les suivis ayant des annotations avec des valeurs spécifiques, utilisez le mot clé `annotation[key]` dans une [expression de filtre](#).

Enregistrement de métadonnées avec le SDK X-Ray pour Ruby

Utilisez les métadonnées pour enregistrer des informations sur les segments ou sous-segments qui n'ont pas besoin d'être indexés pour la recherche. Les valeurs des métadonnées peuvent être de type chaîne, valeur numérique, valeur booléenne ou tout objet à même d'être sérialisé en un tableau ou objet JSON.

Pour enregistrer les métadonnées

1. Obtenez une référence au segment ou sous-segment en cours à partir d'`xray_recorder`.

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

or

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

2. Appelez `metadata` avec une clé de type chaîne, une valeur de type booléen, numérique, chaîne ou objet et un espace de noms de type chaîne.

```
my_metadata = {  
  my_namespace: {  
    key: 'value'  
  }  
}
```



```
}  
subsegment.metadata my_metadata
```

Si vous appelez deux fois `metadata` avec la même clé, les valeurs précédemment enregistrées sur le même segment ou sous-segment sont remplacées.

Utilisateur enregistrant IDs avec le SDK X-Ray pour Ruby

Enregistrez les segments utilisateur IDs sur demande pour identifier l'utilisateur qui a envoyé la demande.

Pour enregistrer un utilisateur IDs

1. Obtenez une référence au segment en cours à partir d'`xray_recorder`.

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

2. Définissez le champ utilisateur du segment sur un ID de chaîne de l'utilisateur qui a envoyé la demande.

```
segment.user = 'U12345'
```

Vous pouvez définir l'utilisateur dans vos contrôleurs pour enregistrer l'ID utilisateur dès que votre application commence le traitement d'une demande.

Pour rechercher les suivis d'un ID utilisateur, utilisez le mot clé `user` dans une [expression de filtre](#).

Migration de l'instrumentation X-Ray vers l' OpenTelemetry instrumentation

X-Ray est en train de devenir OpenTelemetry (OTel) sa principale norme d'instrumentation pour le suivi et l'observabilité des applications. Ce changement stratégique s'aligne sur les AWS meilleures pratiques du secteur et offre aux clients une solution plus complète, flexible et tournée vers l'avenir pour répondre à leurs besoins en matière d'observabilité. OpenTelemetry, largement adoptée dans le secteur, permet de suivre les demandes sur divers systèmes, y compris ceux extérieurs AWS qui peuvent ne pas s'intégrer directement à X-Ray.

Ce chapitre fournit des recommandations pour une transition en douceur et souligne l'importance de migrer vers des solutions OpenTelemetry basées pour garantir un support continu et un accès aux dernières fonctionnalités en matière d'instrumentation et d'observabilité des applications.

Il est recommandé de l'adopter OpenTelemetry comme solution d'observabilité pour l'instrumentation de votre application.

Rubriques

- [Compréhension OpenTelemetry](#)
- [Comprendre les OpenTelemetry concepts relatifs à la migration](#)
- [Présentation de la migration](#)
- [Migration de X-Ray Daemon vers un AWS CloudWatch agent ou un collecteur OpenTelemetry](#)
- [Migration vers Java OpenTelemetry](#)
- [Migrer vers OpenTelemetry Go](#)
- [Migrer vers OpenTelemetry Node.js](#)
- [Migrer vers OpenTelemetry .NET](#)
- [Migrer vers OpenTelemetry Python](#)
- [Migrer vers OpenTelemetry Ruby](#)

Compréhension OpenTelemetry

OpenTelemetry est un cadre d'observabilité standard qui fournit des protocoles et des outils standardisés pour la collecte de données de télémétrie. Il propose une approche unifiée pour

l'instrumentation, la génération, la collecte et l'exportation de données de télémétrie telles que les métriques, les journaux et les traces.

Lorsque vous migrez de X-Ray SDKs vers OpenTelemetry, vous bénéficiez des avantages suivants :

- Support amélioré du framework et de l'instrumentation de bibliothèque
- Support pour d'autres langages de programmation
- Capacités d'instrumentation automatique
- Options de configuration d'échantillonnage flexibles
- Collecte unifiée de métriques, de journaux et de traces

Le OpenTelemetry collecteur offre davantage d'options pour les formats de collecte de données et les destinations d'exportation que le daemon X-Ray.

OpenTelemetry soutien dans AWS

AWS propose plusieurs solutions pour travailler avec OpenTelemetry :

- AWS Distro pour OpenTelemetry

Exportez OpenTelemetry les traces sous forme de segments vers X-Ray.

Pour plus d'informations, consultez [AWS Distro for OpenTelemetry](#).

- CloudWatch Signaux d'application

Exportez OpenTelemetry des traces et des métriques personnalisées pour surveiller l'état de santé des applications.

Pour plus d'informations, consultez la section [Utilisation des signaux d'application](#).

- CloudWatch OTel Point final

Exportez OpenTelemetry les traces vers X-Ray à l'aide du point de OTel terminaison HTTP avec une OpenTelemetry instrumentation native.

Pour plus d'informations, consultez la section [Utilisation des OTel points de terminaison](#).

Utilisation OpenTelemetry avec AWS CloudWatch

AWS CloudWatch prend en charge les OpenTelemetry traces via l'instrumentation des applications côté client et les AWS CloudWatch services natifs tels que les signaux d'application, les traces, les cartes, les métriques et les journaux. Pour de plus amples informations, veuillez consulter [OpenTelemetry](#).

Comprendre les OpenTelemetry concepts relatifs à la migration

Le tableau suivant fait correspondre les concepts X-Ray à leurs OpenTelemetry équivalents. La compréhension de ces mappages vous permet de traduire votre instrumentation X-Ray existante en OpenTelemetry :

Concept X-Ray	OpenTelemetry concept
Enregistreur X-Ray	Fournisseur de traceurs et traceurs
Plug-ins de service	Détecteur de ressources
Segment	Espace (serveur)
Sous-segment	Span (hors serveur)
Règles d'échantillonnage aux rayons X	OpenTelemetry Échantillonnage (personnalisable)
Émetteur de rayons X	Span Exporter (personnalisable)
Annotations/Métadonnées	Attributs
Instrumentation de bibliothèque	Instrumentation de bibliothèque
Contexte de X-Ray Trace	Contexte Span
Propagation du contexte de X-Ray Trace	Propagation du contexte de trace du W3C
Échantillonnage de traces par rayons X	OpenTelemetry échantillonnage de traces
N/A	Traitement Span

Concept X-Ray	OpenTelemetry concept
N/A	Bagages
Démon X-Ray	OpenTelemetry Collectionneur

 Note

Pour plus d'informations sur OpenTelemetry les concepts, consultez la [OpenTelemetry documentation](#).

Comparaison des fonctionnalités

Le tableau suivant indique les fonctionnalités prises en charge dans les deux services. Utilisez ces informations pour identifier les lacunes que vous devez combler lors de la migration :

Fonctionnalité	Instrumentation X-Ray	OpenTelemetry instrumentation
Instrumentation de bibliothèque	Pris en charge	Pris en charge
Échantillonnage aux rayons X	Pris en charge	Pris en charge dans OTel Java/.net/GO Pris en charge dans ADOT Java/. NET/Python/Node.js
Propagation du contexte de X-Ray Trace	Pris en charge	Pris en charge
Détection des ressources	Pris en charge	Pris en charge
Annotations de segments	Pris en charge	Pris en charge
Métadonnées du segment	Pris en charge	Pris en charge

Fonctionnalité	Instrumentation X-Ray	OpenTelemetry instrumentation
Instrumentation automatique à code zéro	Pris en charge en Java	Pris en charge en OTel Java/.NET/Python/Node.js Pris en charge dans ADOT Java/.NET/Python/Node.js
Création manuelle de traces	Pris en charge	Pris en charge

Configuration et configuration du suivi

Pour créer des traces dans OpenTelemetry, vous avez besoin d'un traceur. Vous obtenez un traceur en initialisant un fournisseur de traceurs dans votre application. Cela est similaire à la façon dont vous utilisez le X-Ray Recorder pour configurer X-Ray et créer des segments et des sous-segments dans un X-Ray Trace.

Note

Le OpenTelemetry Tracer Provider propose davantage d'options de configuration que le X-Ray Recorder.

Comprendre la structure des données de trace

Après avoir compris les concepts de base et les mappages de fonctionnalités, vous pouvez en apprendre davantage sur les détails de mise en œuvre spécifiques tels que la structure des données de trace et l'échantillonnage.

OpenTelemetry utilise des étendues plutôt que des segments et des sous-segments pour structurer les données de suivi. Chaque travée inclut les composants suivants :

- Nom
- Identifiant unique
- Horodatages de début et de fin
- Type d'envergure

- Contexte de Span
- Attributs (métadonnées clé-valeur)
- Événements (journaux horodatés)
- Liens vers d'autres spans
- Informations sur le statut
- Références relatives à l'espace parent

Lorsque vous migrez vers OpenTelemetry, vos travées sont automatiquement converties en segments ou sous-segments X-Ray. Cela garantit que votre expérience de CloudWatch console existante reste inchangée.

Utilisation des attributs span

Le SDK X-Ray propose deux méthodes pour ajouter des données aux segments et sous-segments :

Annotations

Paires clé-valeur indexées pour le filtrage et la recherche

Métadonnées

Paires clé-valeur contenant des données complexes qui ne sont pas indexées pour la recherche

Par défaut, les attributs OpenTelemetry span sont convertis en métadonnées dans les données brutes de X-Ray. Pour convertir des attributs spécifiques en annotations, ajoutez leurs clés à la liste des `aws.xray.annotations` attributs.

- Pour plus d'informations sur OpenTelemetry les concepts, voir [OpenTelemetry Traces](#)
- Pour plus de détails sur la façon dont OpenTelemetry les données sont mappées aux données X-Ray, [OpenTelemetry voir Traduction du modèle de données X-Ray](#)

Détecter les ressources dans votre environnement

OpenTelemetry utilise des détecteurs de ressources pour collecter des métadonnées sur les ressources qui génèrent des données de télémétrie. Ces métadonnées sont stockées sous forme d'attributs de ressource. Par exemple, une entité produisant de la télémétrie peut être un cluster Amazon ECS ou une EC2 instance Amazon, et les attributs de ressources qui peuvent être

enregistrés à partir de ces entités peuvent inclure l'ARN du cluster Amazon ECS ou l'ID d' EC2 instance Amazon.

- Pour plus d'informations sur les types de ressources pris en charge, voir [OpenTelemetry Conventions sémantiques des ressources](#)
- Pour plus d'informations sur les plug-ins du service X-Ray, consultez [Configuration du SDK X-Ray](#)

Gestion des stratégies d'échantillonnage

L'échantillonnage de traces vous aide à gérer les coûts en collectant des données à partir d'un sous-ensemble représentatif de demandes plutôt que de toutes les demandes. Les deux, OpenTelemetry ainsi que X-Ray, prennent en charge l'échantillonnage, mais l'implémentent différemment.

Note

L'échantillonnage de moins de 100 % des traces réduit vos coûts d'observabilité tout en conservant des informations pertinentes sur les performances de votre application.

OpenTelemetry fournit plusieurs stratégies d'échantillonnage intégrées et vous permet de créer des stratégies personnalisées. Vous pouvez également configurer un X-Ray Remote Sampler dans certains langages du SDK pour utiliser les règles d'échantillonnage X-Ray avec. OpenTelemetry

Les stratégies d'échantillonnage supplémentaires OpenTelemetry sont les suivantes :

- Échantillonnage basé sur les parents — Respecte la décision d'échantillonnage de l'intervalle parent avant d'appliquer des stratégies d'échantillonnage supplémentaires
- Échantillonnage basé sur le ratio d'identification des traces — >Échantillonne de manière aléatoire un pourcentage spécifié de plages
- Échantillonnage de la queue — Applique les règles d'échantillonnage pour compléter les traces dans le OpenTelemetry collecteur
- Échantillonneurs personnalisés — Implémentez votre propre logique d'échantillonnage à l'aide de l'interface d'échantillonnage

Pour plus d'informations sur les règles d'échantillonnage X-Ray, voir [Règles d'échantillonnage dans la console X-Ray](#)

Pour plus d'informations sur l'échantillonnage de OpenTelemetry la queue, voir [Processeur d'échantillonnage de la queue](#)

Gestion du contexte de suivi

X-Ray SDKs gère le contexte du segment pour gérer correctement les relations parent-enfant entre les segments et les sous-segments d'une trace. OpenTelemetry utilise un mécanisme similaire pour s'assurer que les travées ont la bonne étendue parent. Il stocke et propage les données de suivi dans un contexte de demande. Par exemple, lorsque votre application traite une demande et crée un intervalle de serveur pour représenter cette demande, OpenTelemetry elle stocke l'intervalle de serveur dans le OpenTelemetry contexte afin que, lorsqu'un intervalle enfant est créé, celui-ci puisse faire référence à l'intervalle du contexte en tant que parent.

Propagation du contexte de trace

X-Ray et OpenTelemetry Use utilisent les en-têtes HTTP pour propager le contexte de trace entre les services. Cela vous permet de relier les données de suivi générées par différents services et de conserver les décisions d'échantillonnage.

Le SDK X-Ray propage automatiquement le contexte de trace à l'aide de l'en-tête X-Ray Trace. Lorsqu'un service en appelle un autre, l'en-tête de trace contient le contexte nécessaire pour maintenir les relations parent-enfant entre les traces.

OpenTelemetry prend en charge plusieurs formats d'en-tête de trace pour la propagation du contexte, notamment :

- Contexte de trace du W3C (par défaut)
- En-tête X-Ray Trace
- Autres formats personnalisés

Note

Vous pouvez configurer OpenTelemetry pour utiliser un ou plusieurs formats d'en-tête. Par exemple, utilisez le X-Ray Propagator pour envoyer le contexte de trace aux AWS services qui prennent en charge le X-Ray Propagator.

Configurez et utilisez le X-Ray Propagator pour activer le suivi entre les AWS services. Cela vous permet de propager le contexte de trace aux points de terminaison API Gateway et aux autres services compatibles avec X-Ray.

- Pour plus d'informations sur les en-têtes de trace X-Ray, consultez la section [En-tête de traçage](#) dans le guide du développeur de X-Ray
- Pour plus d'informations sur OpenTelemetry la propagation du [contexte, voir Contexte et propagation du contexte](#) dans la OpenTelemetry documentation

Utilisation de l'instrumentation de bibliothèque

X-Ray et OpenTelemetry Supply Library Instrumentation qui nécessitent un minimum de modifications de code pour ajouter du traçage à vos applications.

X-Ray fournit des fonctionnalités d'instrumentation de bibliothèque. Cela vous permet d'ajouter des instruments X-Ray prédéfinis avec un minimum de modifications du code d'application. Ces instruments prennent en charge des bibliothèques spécifiques telles que le AWS SDK et les clients HTTP, ainsi que des frameworks Web tels que Spring Boot ou Express.js.

OpenTelemetry les bibliothèques d'instrumentation génèrent des étendues détaillées pour vos bibliothèques par le biais de crochets de bibliothèque ou de modifications automatiques du code, nécessitant un minimum de modifications de code.

Pour déterminer si OpenTelemetry Library Instrumentations prend en charge votre bibliothèque, recherchez-la dans le OpenTelemetry registre à [OpenTelemetry Registry](#).

Exporter des traces

X-Ray et OpenTelemetry utilisent différentes méthodes pour exporter les données de trace.

Exportation de X-Ray Trace

Le X-Ray SDKs utilise un émetteur pour envoyer des données de trace :

- Envoie des segments et des sous-segments au X-Ray Daemon
- Utilise le protocole UDP pour des E/S non bloquantes
- Configuré par défaut dans le SDK

OpenTelemetry exportation de traces

OpenTelemetry utilise des Span Exporters configurables pour envoyer des données de suivi :

- Utilise les protocoles http/proto.buf ou grpc
- Les exportations s'étendent vers les points de terminaison surveillés par le OpenTelemetry collecteur ou l'agent CloudWatch
- Permet des configurations d'exportation personnalisées

Traitement et transmission des traces

X-Ray et OpenTelemetry Supply fournissent des composants pour recevoir, traiter et transmettre les données de suivi.

Traitement des traces X-Ray

Le X-Ray Daemon gère le traitement des traces :

- Écoute le trafic UDP provenant de X-Ray SDKs
- Lots, segments et sous-segments
- Télécharge des lots vers le service X-Ray

OpenTelemetry traitement des traces

Le OpenTelemetry Collector gère le traitement des traces :

- Reçoit les traces des services instrumentés
- Traite et modifie éventuellement les données de suivi
- Envoie les traces traitées à différents backends, y compris X-Ray

Note

L' AWS CloudWatch agent peut également recevoir et envoyer OpenTelemetry des traces à X-Ray. Pour plus d'informations, consultez la section [Collecter des métriques et des traces avec OpenTelemetry](#).

Traitement de l'OpenTelemetry envergure (concept spécifique)

OpenTelemetry utilise les processeurs Span pour modifier les spans au fur et à mesure de leur création :

- Permet de lire et de modifier les tranches lors de leur création ou de leur achèvement
- Active une logique personnalisée pour la gestion de l'intervalle

OpenTelemetry Bagages (concept spécifique)

OpenTelemetry La fonction Baggage permet la propagation de données clé-valeur :

- Permet de transmettre des données arbitraires à côté du contexte de trace
- Utile pour diffuser des informations spécifiques à une application au-delà des limites des services

Pour plus d'informations sur le OpenTelemetry Collector, voir [OpenTelemetry Collector](#)

Pour plus d'informations sur les concepts de X-Ray, voir les [concepts de X-Ray](#) dans le X-Ray Developer Guide

Présentation de la migration

Cette section fournit une vue d'ensemble des modifications de code requises pour la migration. La liste ci-dessous contient des instructions spécifiques à la langue et les étapes de migration de X-Ray Daemon.

Important

Pour effectuer une migration complète de l'instrumentation X-Ray vers l' OpenTelemetry instrumentation, vous devez :

1. Remplacez l'utilisation du SDK X-Ray par une solution OpenTelemetry
2. Remplacez le daemon X-Ray par l' CloudWatch agent ou le OpenTelemetry collecteur (avec X-Ray Exporter)

- [Migration vers Java OpenTelemetry](#)
- [Migrer vers OpenTelemetry Go](#)

- [Migrer vers OpenTelemetry Node.js](#)
- [Migrer vers OpenTelemetry .NET](#)
- [Migrer vers OpenTelemetry Python](#)
- [Migrer vers OpenTelemetry Ruby](#)

Recommandations pour les applications nouvelles et existantes

Pour les applications nouvelles et existantes, il est recommandé d'utiliser les solutions suivantes pour activer le suivi dans vos applications :

Instrumentation

- OpenTelemetry SDKs
- AWS Distro pour instrumentation OpenTelemetry

Collecte de données

- OpenTelemetry Collectionneur
- CloudWatch Agent

Après la migration vers des solutions OpenTelemetry basées, votre CloudWatch expérience restera la même. Vous pourrez toujours consulter vos traces dans le même format sur les pages Traces et Trace Map de la CloudWatch console, ou récupérer vos données de trace via le [X-Ray APIs](#).

Modification de la configuration du suivi

Vous devez remplacer la configuration X-Ray par une OpenTelemetry configuration.

Comparaison entre X-Ray et OpenTelemetry configuration

Fonctionnalité	Kit de développement X-Ray	OpenTelemetry
Configurations par défaut	• Échantillonnage centralisé par rayons X • Propagation du contexte X-Ray Trace • Exportation de traces vers X-Ray Daemon	• Exportation de traces vers OpenTelemetry Collector ou CloudWatch Agent (HTTP/GRPC) • Propagation du contexte de trace du W3C

Fonctionnalité	Kit de développement X-Ray	OpenTelemetry
Configurations manuelles	• Règles d'échantillonnage locales • Plug-ins de détection de ressources	• X-Ray Sampling (peut ne pas être disponible dans toutes les langues) • Détection des ressources • Propagation du contexte X-Ray Trace

Modifications apportées aux instruments de bibliothèque

Mettez à jour votre code pour utiliser OpenTelemetry Library Instrumentation au lieu de X-Ray Library Instrumentation pour le AWS SDK, les clients HTTP, les frameworks Web et d'autres bibliothèques. Cela génère OpenTelemetry des Traces au lieu de X-Ray Traces.

Note

Les modifications de code varient en fonction de la langue et de la bibliothèque. Reportez-vous aux guides de migration spécifiques aux langues pour obtenir des instructions détaillées.

Modifications apportées à l'instrumentation de l'environnement Lambda

Pour l'utiliser OpenTelemetry dans vos fonctions Lambda, choisissez l'une des options de configuration suivantes :

1. Utilisez une couche Lambda d'auto-instrumentation :

- Couche [Lambda des signaux CloudWatch d'application](#) (recommandée)

Note

Pour utiliser uniquement le traçage, définissez la variable d'environnement Lambda.
OTEL_AWS_APPLICATION_SIGNALS_ENABLED=false

- [AWS couche Lambda gérée pour ADOT](#)

2. Configuration manuelle OpenTelemetry de votre fonction Lambda :

- Configurer un processeur Simple Span avec un X-Ray UDP Span Exporter
- Configuration d'un propagateur X-Ray Lambda

Création manuelle de données de trace

Remplacez les segments et sous-segments de X-Ray par des OpenTelemetry Spans :

- Utiliser un OpenTelemetry traceur pour créer des spans
- Ajouter des attributs à Spans (équivalent aux métadonnées et aux annotations de X-Ray)

Important

Lorsqu'il est envoyé à X-Ray :

- Les étendues de serveurs sont converties en segments X-Ray
- Les autres spans sont convertis en sous-segments X-Ray
- Les attributs sont convertis en métadonnées par défaut

Pour convertir un attribut en annotation, ajoutez sa clé à la liste des `aws.xray.annotations` attributs. Pour plus d'informations, voir [Activer les annotations X-Ray personnalisées](#).

Migration de X-Ray Daemon vers un AWS CloudWatch agent ou un collecteur OpenTelemetry

Vous pouvez utiliser l' CloudWatch agent ou le OpenTelemetry collecteur pour recevoir les traces de vos applications instrumentées et les envoyer à X-Ray.

Note

La version 1.300025.0 et les versions ultérieures de l' CloudWatch agent peuvent collecter des traces. OpenTelemetry L'utilisation de l' CloudWatch agent au lieu du X-Ray Daemon réduit le nombre d'agents à gérer. Pour plus d'informations, consultez la section [Collecte de métriques, de journaux et de traces avec l' CloudWatch agent](#).

Sections

- [Migration sur Amazon EC2 ou sur des serveurs sur site](#)
- [Migration sur Amazon ECS](#)
- [Migration sur Elastic Beanstalk](#)

Migration sur Amazon EC2 ou sur des serveurs sur site

Important

Arrêtez le processus X-Ray Daemon avant d'utiliser l' CloudWatch agent ou le OpenTelemetry collecteur pour éviter les conflits de ports.

Configuration existante de X-Ray Daemon

Installation du daemon

Votre utilisation actuelle de X-Ray Daemon a été installée à l'aide de l'une des méthodes suivantes :

Installation manuelle

Téléchargez et exécutez le fichier exécutable depuis le bucket Amazon S3 du daemon X-Ray.

Installation automatique

Utilisez ce script pour installer le daemon lors du lancement d'une instance :

```
#!/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
```

Configuration du démon

Votre utilisation actuelle de X-Ray Daemon a été configurée à l'aide de l'une des méthodes suivantes :

- Arguments de ligne de commande

- Fichier de configuration (`xray-daemon.yaml`)

Exemple Utilisation d'un fichier de configuration

```
./xray -c ~/xray-daemon.yaml
```

Exécution du démon

Votre utilisation actuelle de X-Ray Daemon a été lancée avec la commande suivante :

```
~/xray-daemon$ ./xray -o -n us-east-1
```

Supprimer le daemon

Pour supprimer le Daemon X-Ray de votre EC2 instance Amazon :

1. Arrêtez le service daemon :

```
systemctl stop xray
```

2. Supprimez le fichier de configuration :

```
rm ~/path/to/xray-daemon.yaml
```

3. S'il est configuré, supprimez le fichier journal :

Note

L'emplacement du fichier journal dépend de votre configuration :

- Configuration de la ligne de commande : `/var/log/xray-daemon.log`
- Fichier de configuration : Vérifiez le `LogPath` réglage

Configuration de l' CloudWatch agent

Installation de l'agent

Pour les instructions d'installation, reportez-vous à [la section Installation de l' CloudWatch agent sur un serveur local](#).

Configuration de l'agent

1. Créez un fichier de configuration pour activer la collecte de traces. Pour plus d'informations, consultez la section [Création du fichier de configuration de l' CloudWatch agent](#).
2. Configurez les autorisations IAM :
 - Attachez un rôle IAM ou spécifiez les informations d'identification de l'agent. Pour plus d'informations, consultez la section [Configuration des rôles IAM](#).
 - Assurez-vous que le rôle ou les informations d'identification incluent `!xray:PutTraceSegments` autorisation.

Démarrage de l'agent

Pour obtenir des instructions sur le démarrage de l'agent, reportez-vous à la section [Démarrage de l' CloudWatch agent à l'aide de la ligne de commande](#).

Configuration du OpenTelemetry collecteur

Installation du collecteur

Téléchargez et installez le OpenTelemetry collecteur adapté à votre système d'exploitation. Pour obtenir des instructions, reportez-vous à la section [Installation du collecteur](#).

Configuration du collecteur

Configurez les composants suivants dans votre collecteur :

- extension `awsproxy`

Nécessaire pour l'échantillonnage aux rayons X

- OTEL récepteurs

Recueille les traces de votre application

- exportateur de rayons X

Envoie des traces à X-Ray

Exemple Configuration du collecteur d'échantillons — `otel-collector-config.yaml`

```
extensions:
```

```
awsproxy:
  endpoint: 127.0.0.1:2000
health_check:

receivers:
  otlp:
    protocols:
      grpc:
        endpoint: 127.0.0.1:4317
      http:
        endpoint: 127.0.0.1:4318

processors:
  batch:

exporters:
  awsxray:
    region: 'us-east-1'

service:
  pipelines:
    traces:
      receivers: [otlp]
      exporters: [awsxray]
  extensions: [awsproxy, health_check]
```

Important

Configurez AWS les informations d'identification avec `l'xray:PutTraceSegments` autorisation. Pour plus d'informations, consultez la section [Spécification des informations d'identification](#).

Démarrage du collecteur

Lancez le collecteur avec votre fichier de configuration :

```
otelcol --config=otel-collector-config.yaml
```

Migration sur Amazon ECS

⚠ Important

Votre rôle de tâche doit être `xray:PutTraceSegments` autorisé pour tous les collecteurs que vous utilisez.

Arrêtez tout conteneur X-Ray Daemon existant avant d'exécuter l' CloudWatch agent ou le conteneur OpenTelemetry collecteur sur le même hôte afin d'éviter les conflits de ports.

Utilisation de l' CloudWatch agent

1. Obtenez l'image Docker dans la galerie [publique Amazon ECR](#).
2. Créez un fichier de configuration nommé `cw-agent-otel.json` :

```
{
  "traces": {
    "traces_collected": {
      "xray": {
        "tcp_proxy": {
          "bind_address": "0.0.0.0:2000"
        }
      },
      "otlp": {
        "grpc_endpoint": "0.0.0.0:4317",
        "http_endpoint": "0.0.0.0:4318"
      }
    }
  }
}
```

3. Stockez la configuration dans le magasin de paramètres de Systems Manager :

1. Ouvrez le fichier <https://console.aws.amazon.com/systems-manager/>
2. Choisissez Créer un paramètre
3. Entrez les valeurs suivantes :
 - Nom : `/ecs/cwagent/otel-config`
 - Niveau : Standard
 - Type : String

- Type de données : Texte
- Valeur : [Collez la configuration cw-agent-otel .json ici]

4. Créez une définition de tâche en utilisant le mode réseau en pont :

Dans votre définition de tâche, la configuration dépend du mode de mise en réseau que vous utilisez. La mise en réseau par pont est le mode par défaut et peut être utilisé dans votre VPC par défaut. Dans un réseau en pont, définissez la variable d'OTEL_EXPORTER_OTLP_TRACES_ENDPOINT pour indiquer au OpenTelemetry SDK quels sont le point de terminaison et le port de l' CloudWatch agent. Vous devez également créer un lien entre le conteneur de votre application et le conteneur Collector pour que les traces soient envoyées depuis le OpenTelemetry SDK de votre application vers le conteneur Collector.

Exemple CloudWatch définition des tâches de l'agent

```
{
  "containerDefinitions": [
    {
      "name": "cwagent",
      "image": "public.ecr.aws/cloudwatch-agent/cloudwatch-agent:latest",
      "portMappings": [
        {
          "containerPort": 4318,
          "hostPort": 4318,
          "protocol": "tcp"
        },
        {
          "containerPort": 4317,
          "hostPort": 4317,
          "protocol": "tcp"
        },
        {
          "containerPort": 2000,
          "hostPort": 2000,
          "protocol": "tcp"
        }
      ],
      "secrets": [
        {
          "name": "CW_CONFIG_CONTENT",
          "valueFrom": "/ecs/cwagent/otel-config"
        }
      ]
    }
  ]
}
```

```
    ],
    {
      "name": "application",
      "image": "APPLICATION_IMAGE",
      "links": ["cwagent"],
      "environment": [
        {
          "name": "OTEL_EXPORTER_OTLP_TRACES_ENDPOINT",
          "value": "http://cwagent:4318/v1/traces"
        }
      ]
    }
  ]
}
```

Pour plus d'informations, consultez [Déploiement de l' CloudWatch agent pour collecter des métriques EC2 au niveau de l'instance Amazon sur Amazon ECS](#).

Utilisation du OpenTelemetry collecteur

1. Obtenez l'image Docker `otel/opentelemetry-collector-contrib` depuis [Docker Hub](#).
2. Créez un fichier de configuration appelé `otel-collector-config.yaml` en utilisant le même contenu que celui indiqué dans la section Amazon sur la EC2 configuration du collecteur, mais mettez à jour les points de terminaison à utiliser `0.0.0.0` au lieu de `127.0.0.1`.
3. Pour utiliser cette configuration dans Amazon ECS, vous pouvez la stocker dans le magasin de paramètres Systems Manager. Accédez d'abord à la console Systems Manager Parameter Store, puis choisissez `Create new parameter`. Créez un nouveau paramètre avec les informations suivantes :
 - Nom : `/ecs/otel/config` (ce nom sera référencé dans la définition de tâche pour le collecteur)
 - Niveau : `Standard`
 - Type : `String`
 - Type de données : `Texte`
 - Valeur : [Collez la configuration `otel-collector-config.yaml` ici]
4. Créez une définition de tâche pour déployer le OpenTelemetry collecteur en utilisant le mode réseau en pont comme exemple.

Dans la définition de la tâche, la configuration dépend du mode réseau que vous utilisez. La mise en réseau par pont est le mode par défaut et peut être utilisé dans votre VPC par défaut. Dans un réseau en pont, définissez la variable d'OTEL_EXPORTER_OTLP_TRACES_ENDPOINT en environnement pour indiquer au OpenTelemetry SDK quels sont le point de terminaison et le port du OpenTelemetry collecteur. Vous devez également créer un lien entre le conteneur de votre application et le conteneur Collector pour que les traces soient envoyées depuis le OpenTelemetry SDK de votre application vers le conteneur Collector.

Exemple OpenTelemetry définition de la tâche du collecteur

```
{
  "containerDefinitions": [
    {
      "name": "otel-collector",
      "image": "otel/opentelemetry-collector-contrib",
      "portMappings": [
        {
          "containerPort": 2000,
          "hostPort": 2000
        },
        {
          "containerPort": 4317,
          "hostPort": 4317
        },
        {
          "containerPort": 4318,
          "hostPort": 4318
        }
      ],
      "command": [
        "--config",
        "env:SSM_CONFIG"
      ],
      "secrets": [
        {
          "name": "SSM_CONFIG",
          "valueFrom": "/ecs/otel/config"
        }
      ]
    }
  ],
}
```

```
{
  "name": "application",
  "image": "APPLICATION_IMAGE",
  "links": ["otel-collector"],
  "environment": [
    {
      "name": "OTEL_EXPORTER_OTLP_TRACES_ENDPOINT",
      "value": "http://otel-collector:4318/v1/traces"
    }
  ]
}
```

Migration sur Elastic Beanstalk

Important

Arrêtez le processus X-Ray Daemon avant d'utiliser l' CloudWatch agent pour éviter les conflits de ports.

Votre intégration X-Ray Daemon existante a été activée à l'aide de la console Elastic Beanstalk ou en configurant X-Ray Daemon dans le code source de votre application à l'aide d'un fichier de configuration.

Utilisation de l' CloudWatch agent

Sur la plateforme Amazon Linux 2, configurez l' CloudWatch agent à l'aide d'un fichier `.ebextensions` de configuration :

1. Créez un répertoire nommé `.ebextensions` à la racine de votre projet
2. Créez un fichier nommé `cloudwatch.config` dans le `.ebextensions` répertoire avec le contenu suivant :

```
files:
  "/opt/aws/amazon-cloudwatch-agent/etc/config.json":
    mode: "0644"
    owner: root
```



```
group: root
content: |
  {
    "traces": {
      "traces_collected": {
        "otlp": {
          "grpc_endpoint": "12.0.0.1:4317",
          "http_endpoint": "12.0.0.1:4318"
        }
      }
    }
  }
}

container_commands:
  start_agent:
    command: /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a
    append-config -c file:/opt/aws/amazon-cloudwatch-agent/etc/config.json -s
```

3. Incluez le `.ebextensions` répertoire dans le bundle de sources de votre application lors du déploiement

Pour plus d'informations sur les fichiers de configuration d'Elastic Beanstalk, [consultez la section Personnalisation avancée](#) de l'environnement avec les fichiers de configuration.

Migration vers Java OpenTelemetry

Cette section fournit des conseils sur la migration du SDK X-Ray vers le SDK pour les OpenTelemetry applications Java.

Sections

- [Solution d'instrumentation automatique à code zéro](#)
- [Solutions d'instrumentation manuelle avec le SDK](#)
- [Suivi des demandes entrantes \(instrumentation Spring Framework\)](#)
- [AWS Instrumentation du SDK v2](#)
- [Instrumentation des appels HTTP sortants](#)
- [Support d'instrumentation pour d'autres bibliothèques](#)
- [Création manuelle de données de trace](#)
- [Instrumentation Lambda](#)

Solution d'instrumentation automatique à code zéro

With X-Ray Java agent

Pour activer l'agent Java X-Ray, les arguments JVM de votre application devaient être modifiés.

```
-javaagent:/path-to-disco/disco-java-agent.jar=pluginPath=/path-to-disco/disco-plugins
```

With OpenTelemetry-based Java agent

Pour utiliser des agents Java OpenTelemetry basés sur Java.

- Utilisez l'agent Java AWS Distro for OpenTelemetry (ADOT) Auto-Instrumentation pour l'instrumentation automatique avec l'agent ADOT Java. Pour plus d'informations, voir [Instrumentation automatique pour les traces et les métriques avec l'agent Java](#). Si vous souhaitez uniquement le suivi, désactivez la variable d'OTEL_METRICS_EXPORTER=none environnement pour exporter les métriques depuis l'agent Java.

(Facultatif) Vous pouvez également activer les signaux CloudWatch d'application lorsque vous instrumentez automatiquement vos applications à l' AWS aide de l'instrumentation automatique ADOT Java pour surveiller l'état actuel des applications et suivre les performances des applications à long terme. Application Signals fournit une vue unifiée et centrée sur les applications de vos applications, services et dépendances, et permet de surveiller et de trier l'état de santé des applications. Pour plus d'informations, veuillez consulter la rubrique [Applications Signals](#).

- Utilisez l'agent OpenTelemetry Java pour l'instrumentation automatique. Pour plus d'informations, consultez la section [Instrumentation à code zéro avec l'agent Java](#).

Solutions d'instrumentation manuelle avec le SDK

Tracing setup with X-Ray SDK

Pour instrumenter votre code avec le SDK X-Ray pour Java, il fallait d'abord configurer AWSXRay la classe avec des plug-ins de service et des règles d'échantillonnage locales, puis utiliser un enregistreur fourni.

```
static { AWS XRayRecorderBuilder builder = AWS
XRayRecorderBuilder.standard().withPlugin(new EC2Plugin()).withPlugin(new
ECSPPlugin()); AWS XRay.setGlobalRecorder(builder.build());
}
```

Tracing setup with OpenTelemetry SDK

Les dépendances suivantes sont requises.

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>io.opentelemetry</groupId>
      <artifactId>opentelemetry-bom</artifactId>
      <version>1.49.0</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
    <dependency>
      <groupId>io.opentelemetry.instrumentation</groupId>
      <artifactId>opentelemetry-instrumentation-bom</artifactId>
      <version>2.15.0</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
<dependencies>
  <dependency>
    <groupId>io.opentelemetry</groupId>
    <artifactId>opentelemetry-sdk</artifactId>
  </dependency>
  <dependency>
    <groupId>io.opentelemetry</groupId>
    <artifactId>opentelemetry-api</artifactId>
  </dependency>
  <dependency>
    <groupId>io.opentelemetry.semconv</groupId>
    <artifactId>opentelemetry-semconv</artifactId>
  </dependency>
  <dependency>
    <groupId>io.opentelemetry</groupId>
    <artifactId>opentelemetry-exporter-otlp</artifactId>
```

```

    </dependency>
    <dependency>
      <groupId>io.opentelemetry.contrib</groupId>
      <artifactId>opentelemetry-aws-xray</artifactId>
      <version>1.46.0</version>
    </dependency>
    <dependency>
      <groupId>io.opentelemetry.contrib</groupId>
      <artifactId>opentelemetry-aws-xray-propagator</artifactId>
      <version>1.46.0-alpha</version>
    </dependency>
    <dependency>
      <groupId>io.opentelemetry.contrib</groupId>
      <artifactId>opentelemetry-aws-resources</artifactId>
      <version>1.46.0-alpha</version>
    </dependency>
  </dependencies>

```

Configurez le OpenTelemetry SDK en instanciant un objet `TracerProvider` et enregistrez globalement un objet `OpenTelemetrySdk`. Configurez les composants suivants :

- Un exportateur OTLP Span (par exemple, `OtlpGrpcSpanExporter`) - Nécessaire pour exporter des traces vers l' CloudWatch agent ou OpenTelemetry le collecteur
- Un propagateur de AWS rayons X : nécessaire pour propager le contexte de trace aux AWS services intégrés à X-Ray
- Un échantillonneur à distance à AWS rayons X : obligatoire si vous devez effectuer des demandes d'échantillonnage à l'aide des règles d'échantillonnage par rayons X
- Détecteurs de ressources (par exemple, `EcsResource` ou `Ec2Resource`) : détectent les métadonnées de l'hôte exécutant votre application

```

import io.opentelemetry.api.common.Attributes;
import io.opentelemetry.context.propagation.ContextPropagators;
import io.opentelemetry.contrib.aws.resource.Ec2Resource;
import io.opentelemetry.contrib.aws.resource.EcsResource;
import io.opentelemetry.contrib.awsxray.AwsXrayRemoteSampler;
import io.opentelemetry.contrib.awsxray.propagator.AwsXrayPropagator;
import io.opentelemetry.exporter.otlp.trace.OtlpGrpcSpanExporter;
import io.opentelemetry.sdk.OpenTelemetrySdk;
import io.opentelemetry.sdk.resources.Resource;
import io.opentelemetry.sdk.trace.SdkTracerProvider;
import io.opentelemetry.sdk.trace.export.BatchSpanProcessor;

```

```
import io.opentelemetry.sdk.trace.samplers.Sampler;
import static io.opentelemetry.semconv.ServiceAttributes.SERVICE_NAME;

// ...

private static final Resource otelResource =
    Resource.create(Attributes.of(SERVICE_NAME, "YOUR_SERVICE_NAME"))
        .merge(EcsResource.get())
        .merge(Ec2Resource.get());
private static final SdkTracerProvider sdkTracerProvider =
    SdkTracerProvider.builder()
        .addSpanProcessor(BatchSpanProcessor.create(
            OtlpGrpcSpanExporter.getDefault()
        ))
        .addResource(otelResource)
        .setSampler(Sampler.parentBased(
            AwsXrayRemoteSampler.newBuilder(otelResource).build()
        ))
        .build();
// Globally registering a TracerProvider makes it available throughout the
// application to create as many Tracers as needed.
private static final OpenTelemetrySdk openTelemetry =
    OpenTelemetrySdk.builder()
        .setTracerProvider(sdkTracerProvider)

        .setPropagators(ContextPropagators.create(AwsXrayPropagator.getInstance()))
        .buildAndRegisterGlobal();
```

Suivi des demandes entrantes (instrumentation Spring Framework)

With X-Ray SDK

Pour plus d'informations sur l'utilisation du SDK X-Ray avec le framework Spring pour instrumenter votre application, consultez [AOP with Spring et le SDK X-Ray](#) pour Java. Pour activer l'AOP au printemps, procédez comme suit.

1. [Configurez Spring](#)
2. [Ajouter un filtre de suivi à votre application](#)
3. [Annotez votre code ou implémentez une interface](#)
4. [Activez X-Ray dans votre application](#)

With OpenTelemetry SDK

OpenTelemetry fournit des bibliothèques d'instrumentation pour collecter des traces pour les demandes entrantes pour les applications Spring Boot. Pour activer l'instrumentation Spring Boot avec une configuration minimale, incluez la dépendance suivante.

```
<dependency>
  <groupId>io.opentelemetry.instrumentation</groupId>
  <artifactId>opentelemetry-spring-boot-starter</artifactId>
</dependency>
```

Pour plus d'informations sur la façon d'activer et de configurer l'instrumentation Spring Boot pour votre OpenTelemetry installation, consultez OpenTelemetry [Getting started](#).

Using OpenTelemetry-based Java agents

La méthode recommandée par défaut pour instrumenter les applications Spring Boot consiste à utiliser l'[agent OpenTelemetry Java](#) avec une instrumentation en bytecode, qui fournit également davantage d'out-of-the-box instrumentations et de configurations par rapport à l'utilisation directe du SDK. Pour commencer, voir [Solution d'instrumentation automatique à code zéro](#).

AWS Instrumentation du SDK v2

With X-Ray SDK

Le SDK X-Ray pour Java peut automatiquement instrumenter AWS tous les clients du SDK v2 lorsque vous avez ajouté `aws-xray-recorder-sdk-aws-sdk-v2-instrumentor` le sous-module dans votre build.

Pour instrumenter les appels clients individuels aux AWS services en aval avec AWS SDK for Java 2.2 et versions ultérieures, `aws-xray-recorder-sdk-aws-sdk-v2-instrumentor` le module de votre configuration de compilation a été exclu et `aws-xray-recorder-sdk-aws-sdk-v2` le module a été inclus. Les clients individuels ont été instrumentés en les configurant avec un `TracingInterceptor`.

```
import com.amazonaws.xray.interceptors.TracingInterceptor;
import software.amazon.awssdk.core.client.config.ClientOverrideConfiguration
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
//...

public class MyModel {
```

```
private DynamoDbClient client = DynamoDbClient.builder()
    .region(Region.US_WEST_2)
    .overrideConfiguration(
        ClientOverrideConfiguration.builder()
            .addExecutionInterceptor(new TracingInterceptor())
            .build()
    )
    .build();
//...
```

With OpenTelemetry SDK

Pour instrumenter automatiquement tous les clients du AWS SDK, ajoutez le `opentelemetry-aws-sdk-2.2-autoconfigure` sous-module.

```
<dependency>
    <groupId>io.opentelemetry.instrumentation</groupId>
    <artifactId>opentelemetry-aws-sdk-2.2-autoconfigure</artifactId>
    <version>2.15.0-alpha</version>
    <scope>runtime</scope>
</dependency>
```

Pour instrumenter des clients AWS SDK individuels, ajoutez le `opentelemetry-aws-sdk-2.2` sous-module.

```
<dependency>
    <groupId>io.opentelemetry.instrumentation</groupId>
    <artifactId>opentelemetry-aws-sdk-2.2</artifactId>
    <version>2.15.0-alpha</version>
    <scope>compile</scope>
</dependency>
```

Enregistrez ensuite un intercepteur lors de la création d'un client AWS SDK.

```
import io.opentelemetry.instrumentation.awssdk.v2_2.AwsSdkTelemetry;

// ...

AwsSdkTelemetry telemetry = AwsSdkTelemetry.create(openTelemetry);
private final S3Client S3_CLIENT = S3Client.builder()
    .overrideConfiguration(ClientOverrideConfiguration.builder()
        .addExecutionInterceptor(telemetry.newExecutionInterceptor())
```

```
.build())
.build();
```

Instrumentation des appels HTTP sortants

With X-Ray SDK

Pour instrumenter les requêtes HTTP sortantes avec X-Ray, le SDK X-Ray pour Java `HttpClient` était requis.

```
import com.amazonaws.xray.proxies.apache.http.HttpClientBuilder;
...
public String randomName() throws IOException {
    CloseableHttpClient httpclient = HttpClientBuilder.create().build();
```

With OpenTelemetry SDK

Comme le SDK Java X-Ray, il OpenTelemetry fournit une `ApacheHttpClientTelemetry` classe dotée d'une méthode de création qui permet de créer une instance d'un `HttpClientBuilder` pour fournir des étendues OpenTelemetry basées et une propagation du contexte pour Apache. `HttpClient`

```
<dependency>
    <groupId>io.opentelemetry.instrumentation</groupId>
    <artifactId>opentelemetry-apache-httpclient-5.2</artifactId>
    <version>2.15.0-alpha</version>
    <scope>compile</scope>
</dependency>
```

Voici un exemple de code tiré du [opentelemetry-java-instrumentation](https://opentelemetry.io/docs/instrumentation/java/httpclient/). Le client HTTP fourni par `newHttpClient()` générera des traces pour les requêtes exécutées.

```
import io.opentelemetry.api.OpenTelemetry;
import
    io.opentelemetry.instrumentation.apachehttpclient.v5_2.ApacheHttpClientTelemetry;
import org.apache.hc.client5.http.classic.HttpClient;
import org.apache.hc.client5.http.impl.classic.HttpClientBuilder;

public class ApacheHttpClientConfiguration {
```



```
private OpenTelemetry openTelemetry;

public ApacheHttpClientConfiguration(OpenTelemetry openTelemetry) {
    this.openTelemetry = openTelemetry;
}

// creates a new http client builder for constructing http clients with open
telemetry instrumentation
public HttpClientBuilder createBuilder() {
    return
ApacheHttpClientTelemetry.builder(openTelemetry).build().newHttpClientBuilder();
}

// creates a new http client with open telemetry instrumentation
public HttpClient newHttpClient() {
    return ApacheHttpClientTelemetry.builder(openTelemetry).build().newHttpClient();
}
}
```

Support d'instrumentation pour d'autres bibliothèques

Retrouvez la liste complète des instrumentations de bibliothèque prises en charge pour OpenTelemetry Java dans son GitHub référentiel d'instrumentation respectif, sous [Bibliothèques, frameworks, serveurs d'applications et JVMs](#).

Vous pouvez également effectuer une recherche OpenTelemetry dans le registre pour savoir si l'instrumentation est OpenTelemetry compatible. Pour commencer la recherche, consultez la section [Registre](#).

Création manuelle de données de trace

With X-Ray SDK

Avec le SDK X-Ray, les `beginSubsegment` méthodes `beginSegment` et sont nécessaires pour créer manuellement des segments et des sous-segments X-Ray.

```
Segment segment = xrayRecorder.beginSegment("ManualSegment");
segment.putAnnotation("annotationKey", "annotationValue");
segment.putMetadata("metadataKey", "metadataValue");

try {
```

```
        Subsegment subsegment =
xrayRecorder.beginSubsegment("ManualSubsegment");
        subsegment.putAnnotation("key", "value");

        // Do something here

    } catch (Exception e) {
        subsegment.addException(e);
    } finally {
        xrayRecorder.endSegment();
    }
}
```

With OpenTelemetry SDK

Vous pouvez utiliser des intervalles personnalisés pour surveiller les performances des activités internes qui ne sont pas capturées par les bibliothèques d'instrumentation. Notez que seuls les serveurs de type span sont convertis en segments X-Ray, tous les autres serveurs sont convertis en sous-segments X-Ray.

Tout d'abord, vous devez créer un traceur afin de générer des spans, que vous pouvez obtenir grâce à la `openTelemetry.getTracer` méthode. Cela fournira une instance Tracer à partir de `TracerProvider` celle qui a été enregistrée globalement dans l'[Solutions d'instrumentation manuelle avec le SDK](#) exemple. Vous pouvez créer autant d'instances de Tracer que nécessaire, mais il est courant d'avoir un seul Tracer pour l'ensemble d'une application.

```
Tracer tracer = openTelemetry.getTracer("my-app");
```

Vous pouvez utiliser le Tracer pour créer des travées.

```
import io.opentelemetry.api.common.AttributeKey;
import io.opentelemetry.api.trace.Span;
import io.opentelemetry.api.trace.SpanKind;
import io.opentelemetry.api.trace.Tracer;
import io.opentelemetry.context.Scope;

...

// SERVER span will become an X-Ray segment
Span span = tracer.spanBuilder("get-token")
    .setKind(SpanKind.SERVER)
    .setAttribute("key", "value")
    .startSpan();
```

```
try (Scope ignored = span.makeCurrent()) {

    span.setAttribute("metadataKey", "metadataValue");
    span.setAttribute("annotationKey", "annotationValue");

    // The following ensures that "annotationKey: annotationValue" is an annotation in
    // X-Ray raw data.
    span.setAttribute(AttributeKey.stringArrayKey("aws.xray.annotations"),
        List.of("annotationKey"));

    // Do something here
}

span.end();
```

Le type par défaut des travées est INTERNAL.

```
// Default span of type INTERNAL will become an X-Ray subsegment
Span span = tracer.spanBuilder("process-header")
    .startSpan();
try (Scope ignored = span.makeCurrent()) {
    doProcessHeader();
}
```

Ajouter des annotations et des métadonnées aux traces avec OpenTelemetry le SDK

Dans l'exemple ci-dessus, la `setAttribute` méthode est utilisée pour ajouter des attributs à chaque intervalle. Par défaut, tous les attributs de span seront convertis en métadonnées dans les données brutes de X-Ray. Pour garantir qu'un attribut est converti en annotation et non en métadonnées, l'exemple ci-dessus ajoute la clé de cet attribut à la liste de `aws.xray.annotationsattribut`. [Pour plus d'informations, voir Activer les annotations et les métadonnées X-Ray personnalisées.](#)

Avec des agents Java OpenTelemetry basés sur Java

Si vous utilisez l'agent Java pour instrumenter automatiquement votre application, vous devez effectuer une instrumentation manuelle dans votre application. Par exemple, pour coder les instruments dans l'application pour les sections qui ne sont couvertes par aucune bibliothèque d'auto-instrumentation.

Pour effectuer une instrumentation manuelle avec l'agent, vous devez utiliser l'`opentelemetry-api` artefact. La version de l'artefact ne peut pas être plus récente que la version de l'agent.

```
import io.opentelemetry.api.GlobalOpenTelemetry;
import io.opentelemetry.api.trace.Span;

// ...

Span parentSpan = Span.current();
Tracer tracer = GlobalOpenTelemetry.getTracer("my-app");
Span span = tracer.spanBuilder("my-span-name")
    .setParent(io.opentelemetry.context.Context.current().with(parentSpan))
    .startSpan();
span.end();
```

Instrumentation Lambda

With X-Ray SDK

À l'aide du SDK X-Ray, une fois qu'Active Tracing est activé sur votre Lambda, aucune configuration supplémentaire n'est requise pour utiliser le SDK X-Ray. Lambda créera un segment représentant l'invocation du gestionnaire Lambda, et vous pouvez créer des sous-segments ou des bibliothèques d'instruments à l'aide du SDK X-Ray sans aucune configuration supplémentaire.

With OpenTelemetry-based solutions

Couches Lambda à instrumentation automatique — Vous pouvez instrumenter automatiquement votre Lambda AWS avec des couches Lambda vendues à l'aide des solutions suivantes :

- CloudWatch Couche Lambda des signaux d'application (recommandée)

Note

Les signaux d' CloudWatch application sont activés par défaut sur cette couche Lambda, ce qui permet de surveiller les performances et l'état de santé de votre application Lambda en collectant à la fois des métriques et des traces. Pour le suivi uniquement, définissez la variable d'environnement Lambda.

```
OTEL_AWS_APPLICATION_SIGNALS_ENABLED=false
```

- Permet de surveiller les performances et l'état de santé de votre application Lambda
- Collecte à la fois les métriques et les traces par défaut

- AWS couche Lambda gérée pour ADOT Java. Pour plus d'informations, consultez [AWS Distro for OpenTelemetry Lambda Support](#) for Java.

Pour utiliser l'instrumentation manuelle avec la couche d'instrumentation automatique, voir [Solutions d'instrumentation manuelle avec le SDK](#). Pour réduire les démarrages à froid, pensez à utiliser une instrumentation OpenTelemetry manuelle pour générer des OpenTelemetry traces pour votre fonction Lambda.

OpenTelemetry instrumentation manuelle pour AWS Lambda

Prenez en compte le code de fonction Lambda suivant qui effectue un appel Amazon S3 ListBuckets .

```
package example;

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

import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.ListBucketsRequest;
import software.amazon.awssdk.services.s3.model.ListBucketsResponse;
import software.amazon.awssdk.services.s3.model.S3Exception;

public class ListBucketsLambda implements RequestHandler<String, String> {

    private final S3Client S3_CLIENT = S3Client.builder()
        .build();

    @Override
    public String handleRequest(String input, Context context) {
        try {
            ListBucketsResponse response = makeListBucketsCall();
            context.getLogger().log("response: " + response.toString());
            return "Success";
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }

    private ListBucketsResponse makeListBucketsCall() {
        try {
```

```
        ListBucketsRequest listBucketsRequest = ListBucketsRequest.builder()
            .build();
        ListBucketsResponse response = S3_CLIENT.listBuckets(listBucketsRequest);
        return response;
    } catch (S3Exception e) {
        throw new RuntimeException("Failed to call S3 listBuckets" +
            e.awsErrorDetails().errorMessage(), e);
    }
}
```

Voici les dépendances.

```
dependencies {
    implementation('com.amazonaws:aws-lambda-java-core:1.2.3')
    implementation('software.amazon.awssdk:s3:2.28.29')
    implementation('org.slf4j:slf4j-nop:2.0.16')
}
```

Pour instrumenter manuellement votre gestionnaire Lambda et le client Amazon S3, procédez comme suit.

1. Remplacez les classes de fonctions qui implémentent `RequestHandler` (ou `RequestStreamHandler`) par celles qui étendent `TracingRequestHandler` (ou `TracingRequestStreamHandler`).
2. Instanciez un objet `TracerProvider` et enregistrez globalement un `OpenTelemetrySdk` objet. `TracerProvider` Il est recommandé de le configurer avec :
 - a. Un processeur `Span` simple avec un exportateur de `span` UDP X-Ray pour envoyer des traces au point de terminaison UDP X-Ray de Lambda
 - b. Un échantillonneur `ParentBased` toujours actif (par défaut s'il n'est pas configuré)
 - c. Une ressource dont `service.name` est défini sur le nom de la fonction Lambda
 - d. Un propagateur X-Ray Lambda
3. Changez la `handleRequest` méthode `doHandleRequest` et transmettez l'`OpenTelemetrySdk` objet à la classe de base.
4. Instrumentez le client Amazon S3 à l'aide de l'instrumentation du `OpenTemetry AWS SDK` en enregistrant l'intercepteur lors de la création du client.

Vous avez besoin des dépendances OpenTelemetry connexes suivantes.

```
dependencies {
    ...

    implementation("software.amazon.distro.opentelemetry:aws-distro-opentelemetry-xray-udp-span-exporter:0.1.0")

    implementation(platform('io.opentelemetry.instrumentation:opentelemetry-instrumentation-bom:2.14.0'))
    implementation(platform('io.opentelemetry:opentelemetry-bom:1.48.0'))

    implementation('io.opentelemetry:opentelemetry-sdk')
    implementation('io.opentelemetry:opentelemetry-api')
    implementation('io.opentelemetry.contrib:opentelemetry-aws-xray-propagator:1.45.0-alpha')
    implementation('io.opentelemetry.contrib:opentelemetry-aws-resources:1.45.0-alpha')
    implementation('io.opentelemetry.instrumentation:opentelemetry-aws-lambda-core-1.0:2.14.0-alpha')
    implementation('io.opentelemetry.instrumentation:opentelemetry-aws-sdk-2.2:2.14.0-alpha')
}
```

Le code suivant illustre la fonction Lambda après les modifications requises. Vous pouvez créer des travées personnalisées supplémentaires pour compléter les travées fournies automatiquement.

```
package example;

import java.time.Duration;

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

import io.opentelemetry.api.common.Attributes;
import io.opentelemetry.context.propagation.ContextPropagators;
import io.opentelemetry.contrib.aws.resource.LambdaResource;
import io.opentelemetry.contrib.awsxray.propagator.AwsXrayLambdaPropagator;
import io.opentelemetry.instrumentation.awslambdacore.v1_0.TracingRequestHandler;
import io.opentelemetry.instrumentation.awssdk.v2_2.AwsSdkTelemetry;
import io.opentelemetry.sdk.OpenTelemetrySdk;
import io.opentelemetry.sdk.resources.Resource;
import io.opentelemetry.sdk.trace.SdkTracerProvider;
import io.opentelemetry.sdk.trace.export.SimpleSpanProcessor;
import io.opentelemetry.sdk.trace.samplers.Sampler;
```

```

import static io.opentelemetry.semconv.ServiceAttributes.SERVICE_NAME;
import software.amazon.awssdk.core.client.config.ClientOverrideConfiguration;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.ListBucketsRequest;
import software.amazon.awssdk.services.s3.model.ListBucketsResponse;
import software.amazon.awssdk.services.s3.model.S3Exception;
import
    software.amazon.distro.opentelemetry.exporter.xray.udp.trace.AwsXrayUdpSpanExporterBuilder;

public class ListBucketsLambda extends TracingRequestHandler<String, String> {
    private static final Resource lambdaResource = LambdaResource.get();
    private static final SdkTracerProvider sdkTracerProvider =
        SdkTracerProvider.builder()
            .addSpanProcessor(SimpleSpanProcessor.create(
                new AwsXrayUdpSpanExporterBuilder().build()
            ))
            .addResource(
                lambdaResource
                .merge(Resource.create(Attributes.of(SERVICE_NAME,
System.getenv("AWS_LAMBDA_FUNCTION_NAME")))))
            )
            .setSampler(Sampler.parentBased(Sampler.alwaysOn()))
            .build();
    private static final OpenTelemetrySdk openTelemetry =
        OpenTelemetrySdk.builder()
            .setTracerProvider(sdkTracerProvider)

.setPropagators(ContextPropagators.create(AwsXrayLambdaPropagator.getInstance()))
            .buildAndRegisterGlobal();
    private static final AwsSdkTelemetry telemetry =
AwsSdkTelemetry.create(openTelemetry);
    private final S3Client S3_CLIENT = S3Client.builder()
        .overrideConfiguration(ClientOverrideConfiguration.builder()
            .addExecutionInterceptor(telemetry.newExecutionInterceptor())
            .build())
        .build();

    public ListBucketsLambda() {
        super(openTelemetry, Duration.ofMillis(0));
    }

    @Override
    public String doHandleRequest(String input, Context context) {
        try {

```



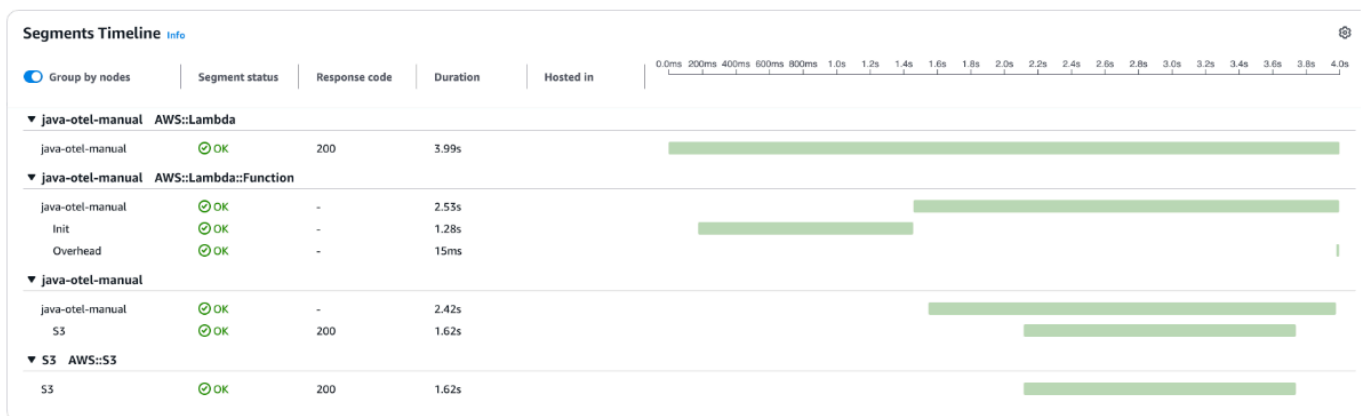
```

        ListBucketsResponse response = makeListBucketsCall();
        context.getLogger().log("response: " + response.toString());
        return "Success";
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
}

private ListBucketsResponse makeListBucketsCall() {
    try {
        ListBucketsRequest listBucketsRequest = ListBucketsRequest.builder()
            .build();
        ListBucketsResponse response = S3_CLIENT.listBuckets(listBucketsRequest);
        return response;
    } catch (S3Exception e) {
        throw new RuntimeException("Failed to call S3 listBuckets" +
            e.awsErrorDetails().errorMessage(), e);
    }
}
}

```

Lorsque vous appelez la fonction Lambda, vous verrez la trace suivante sous Trace Map dans CloudWatch la console.



Migrer vers OpenTelemetry Go

Utilisez les exemples de code suivants pour instrumenter manuellement vos applications Go avec le OpenTelemetry SDK lors de la migration depuis X-Ray.

Instrumentation manuelle avec le SDK

Tracing setup with X-Ray SDK

Lors de l'utilisation du SDK X-Ray pour Go, les plug-ins de service ou les règles d'échantillonnage locales devaient être configurés avant d'instrumenter votre code.

```
func init() {
    if os.Getenv("ENVIRONMENT") == "production" {
        ec2.Init()
    }

    xray.Configure(xray.Config{
        DaemonAddr:      "127.0.0.1:2000",
        ServiceVersion:   "1.2.3",
    })
}
```

Set up tracing with OpenTelemetry SDK

Configurez le OpenTelemetry SDK en instanciant un TracerProvider et en l'enregistrant en tant que fournisseur de traceurs global. Nous vous recommandons de configurer les composants suivants :

- Exportateur de traces OTLP — Nécessaire pour exporter des traces vers l' CloudWatch agent ou OpenTelemetry le collecteur
- X-Ray Propagator — Nécessaire pour propager le contexte de trace aux AWS services intégrés à X-Ray
- X-Ray Remote Sampler : obligatoire pour les demandes d'échantillonnage utilisant les règles d'échantillonnage X-Ray
- Détecteurs de ressources : pour détecter les métadonnées de l'hôte exécutant votre application

```
import (  
    "go.opentelemetry.io/contrib/detectors/aws/ec2"  
    "go.opentelemetry.io/contrib/propagators/aws/xray"  
    "go.opentelemetry.io/contrib/samplers/aws/xray"  
    "go.opentelemetry.io/otel"  
    "go.opentelemetry.io/otel/exporters/otlp/otlptrace/otlptracegrpc"  
    "go.opentelemetry.io/otel/sdk/trace"  
)  
  
func setupTracing() error {  
    ctx := context.Background()  
  
    exporterEndpoint := os.Getenv("OTEL_EXPORTER_OTLP_ENDPOINT")  
    if exporterEndpoint == "" {  
        exporterEndpoint = "localhost:4317"  
    }  
  
    traceExporter, err := otlptracegrpc.New(ctx,  
        otlptracegrpc.WithInsecure(),  
        otlptracegrpc.WithEndpoint(exporterEndpoint))  
    if err != nil {  
        return fmt.Errorf("failed to create OTLP trace exporter: %v", err)  
    }  
  
    remoteSampler, err := xray.NewRemoteSampler(ctx, "my-service-name", "ec2")  
    if err != nil {  
        return fmt.Errorf("failed to create X-Ray Remote Sampler: %v", err)  
    }  
  
    ec2Resource, err := ec2.NewResourceDetector().Detect(ctx)  
    if err != nil {  
        return fmt.Errorf("failed to detect EC2 resource: %v", err)  
    }  
  
    tp := trace.NewTracerProvider(  
        trace.WithSampler(remoteSampler),  
        trace.WithBatcher(traceExporter),  
        trace.WithResource(ec2Resource),  
    )  
  
    otel.SetTracerProvider(tp)  
    otel.SetTextMapPropagator(xray.Propagator{})  
}
```

```
    return nil
}
```

Suivi des demandes entrantes (instrumentation du gestionnaire HTTP)

With X-Ray SDK

Pour instrumenter un gestionnaire HTTP avec X-Ray, la méthode du gestionnaire X-Ray a été utilisée pour générer des segments à l'aide de `NewFixedSegmentNamer`

```
func main() {
    http.Handle("/", xray.Handler(xray.NewFixedSegmentNamer("myApp"),
    http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
        w.Write([]byte("Hello!"))
    })))
    http.ListenAndServe(":8000", nil)
}
```

With OpenTelemetry SDK

Pour instrumenter un gestionnaire HTTP OpenTelemetry, utilisez la OpenTelemetry méthode `newHandler` pour encapsuler le code de votre gestionnaire d'origine.

```
import (
    "go.opentelemetry.io/contrib/instrumentation/net/http/otelhttp"
)

helloHandler := func(w http.ResponseWriter, req *http.Request) {
    ctx := req.Context()
    span := trace.SpanFromContext(ctx)
    span.SetAttributes(attribute.Bool("isHelloHandlerSpan", true),
    attribute.String("attrKey", "attrValue"))

    _, _ = io.WriteString(w, "Hello World!\n")
}
```

```
otelHandler := otelhttp.NewHandler(http.HandlerFunc(helloHandler), "Hello")

http.Handle("/hello", otelHandler)
err = http.ListenAndServe(":8080", nil)
if err != nil {
    log.Fatal(err)
}
```

AWS SDK pour l'instrumentation du SDK Go v2

With X-Ray SDK

Pour instrumenter les AWS demandes sortantes depuis le AWS SDK, vos clients ont été instrumentés comme suit :

```
// Create a segment
ctx, root := xray.BeginSegment(context.TODO(), "AWSSDKV2_Dynamodb")
defer root.Close(nil)

cfg, err := config.LoadDefaultConfig(ctx, config.WithRegion("us-west-2"))
if err != nil {
    log.Fatalf("unable to load SDK config, %v", err)
}
// Instrumenting AWS SDK v2
awsv2.AWSSV2Instrumentor(&cfg.APIOptions)
// Using the Config value, create the DynamoDB client
svc := dynamodb.NewFromConfig(cfg)
// Build the request with its input parameters
_, err = svc.ListTables(ctx, &dynamodb.ListTablesInput{
    Limit: aws.Int32(5),
})
if err != nil {
    log.Fatalf("failed to list tables, %v", err)
}
```

With OpenTelemetry SDK

La prise en charge du suivi des appels du AWS SDK en aval est fournie par OpenTelemetry le AWS SDK for Go v2 Instrumentation. Voici un exemple de suivi d'un appel client S3 :

```

import (
    ...

    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/s3"

    "go.opentelemetry.io/otel"
    oteltrace "go.opentelemetry.io/otel/trace"
    awsConfig "github.com/aws/aws-sdk-go-v2/config"
    "go.opentelemetry.io/contrib/instrumentation/github.com/aws/aws-sdk-go-v2/
otelaws"
)

...

// init aws config
cfg, err := awsConfig.LoadDefaultConfig(ctx)
if err != nil {
    panic("configuration error, " + err.Error())
}

// instrument all aws clients
otelaws.AppendMiddlewares(&.APIOptions)

// Call to S3
s3Client := s3.NewFromConfig(cfg)
input := &s3.ListBucketsInput{}
result, err := s3Client.ListBuckets(ctx, input)
if err != nil {
    fmt.Printf("Got an error retrieving buckets, %v", err)
    return
}

```

Instrumentation des appels HTTP sortants

With X-Ray SDK

Pour instrumenter les appels HTTP sortants avec X-Ray, le `Xray.Client` a été utilisé pour créer une copie d'un client HTTP fourni.

```
myClient := xray.Client(http-client)

resp, err := ctxhttp.Get(ctx, xray.Client(nil), url)
```

With OpenTelemetry SDK

Pour instrumenter les clients HTTP OpenTelemetry, utilisez OpenTelemetry `otelhttp`. `NewTransport` méthode pour encapsuler le `http.DefaultTransport`.

```
import (
    "go.opentelemetry.io/contrib/instrumentation/net/http/otelhttp"
)

// Create an instrumented HTTP client.
httpClient := &http.Client{
    Transport: otelhttp.NewTransport(
        http.DefaultTransport,
    ),
}

req, err := http.NewRequestWithContext(ctx, http.MethodGet, "https://api.github.com/
repos/aws-observability/aws-otel-go/releases/latest", nil)
if err != nil {
    fmt.Printf("failed to create http request, %v\n", err)
}
res, err := httpClient.Do(req)
if err != nil {
    fmt.Printf("failed to make http request, %v\n", err)
}
// Request body must be closed
defer func(Body io.ReadCloser) {
    err := Body.Close()
    if err != nil {
        fmt.Printf("failed to close http response body, %v\n", err)
    }
}(res.Body)
```

Support d'instrumentation pour d'autres bibliothèques

Vous trouverez la liste complète des instrumentations de bibliothèque prises en charge pour les [packages OpenTelemetry Go sous Instrumentation](#).

Vous pouvez également effectuer une recherche OpenTelemetry dans le registre pour savoir si l'instrumentation est OpenTelemetry compatible avec votre bibliothèque sous [Registre](#).

Création manuelle de données de trace

With X-Ray SDK

Avec le SDK X-Ray, les `BeginSubsegment` méthodes `BeginSegment` et étaient nécessaires pour créer manuellement des segments et des sous-segments X-Ray.

```
// Start a segment
ctx, seg := xray.BeginSegment(context.Background(), "service-name")
// Start a subsegment
subCtx, subSeg := xray.BeginSubsegment(ctx, "subsegment-name")

// Add metadata or annotation here if necessary
xray.AddAnnotation(subCtx, "annotationKey", "annotationValue")
xray.AddMetadata(subCtx, "metadataKey", "metadataValue")

subSeg.Close(nil)
// Close the segment
seg.Close(nil)
```

With OpenTelemetry SDK

Utilisez des intervalles personnalisés pour surveiller les performances des activités internes qui ne sont pas capturées par les bibliothèques d'instrumentation. Notez que seules les pages de type `Server` sont converties en segments X-Ray, toutes les autres pages sont converties en sous-segments de X-Ray.

Tout d'abord, vous devez créer un traceur pour générer des spans, que vous pouvez obtenir grâce à la `otel.Tracer` méthode. Cela fournira une instance de `Tracer TracerProvider` qui a été enregistrée globalement dans l'exemple de configuration du suivi. Vous pouvez créer autant

d'instances de Tracer que nécessaire, mais il est courant d'avoir un seul Tracer pour l'ensemble d'une application.

```
tracer := otel.Tracer("application-tracer")
```

```
import (
    ...

    oteltrace "go.opentelemetry.io/otel/trace"
)

...

var attributes = []attribute.KeyValue{
    attribute.KeyValue{Key: "metadataKey", Value:
attribute.StringValue("metadataValue")},
    attribute.KeyValue{Key: "annotationKey", Value:
attribute.StringValue("annotationValue")},
    attribute.KeyValue{Key: "aws.xray.annotations", Value:
attribute.StringSliceValue([]string{"annotationKey"})},
}

ctx := context.Background()

parentSpanContext, parentSpan := tracer.Start(ctx,
"ParentSpan", oteltrace.WithSpanKind(oteltrace.SpanKindServer),
oteltrace.WithAttributes(attributes...))
_, childSpan := tracer.Start(parentSpanContext, "ChildSpan",
oteltrace.WithSpanKind(oteltrace.SpanKindInternal))

// ...

childSpan.End()
parentSpan.End()
```

Ajouter des annotations et des métadonnées aux traces avec OpenTelemetry le SDK

Dans l'exemple ci-dessus, la `WithAttributes` méthode est utilisée pour ajouter des attributs à chaque intervalle. Notez que par défaut, tous les attributs de span sont convertis en métadonnées dans les données brutes de X-Ray. Pour vous assurer qu'un attribut est converti en annotation et non en métadonnées, ajoutez la clé de l'attribut à la liste de l'`aws.xray.annotationsattribut`. Pour plus d'informations, consultez [Enable The Customized X-Ray Annotations](#).

Instrumentation manuelle Lambda

With X-Ray SDK

Avec le SDK X-Ray, une fois qu'Active Tracing a été activé sur votre Lambda, aucune configuration supplémentaire n'était requise pour utiliser le SDK X-Ray. Lambda a créé un segment représentant l'invocation du gestionnaire Lambda, et vous avez créé des sous-segments à l'aide du SDK X-Ray sans aucune configuration supplémentaire.

With OpenTelemetry SDK

Le code de fonction Lambda suivant (sans instrumentation) effectue un ListBuckets appel Amazon S3 et une requête HTTP sortante.

```
package main

import (
    "context"
    "encoding/json"
    "fmt"
    "io"
    "net/http"
    "os"

    "github.com/aws/aws-lambda-go/events"
    "github.com/aws/aws-lambda-go/lambda"
    awsconfig "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/s3"
)

func lambdaHandler(ctx context.Context) (interface{}, error) {
    // Initialize AWS config.
    cfg, err := awsconfig.LoadDefaultConfig(ctx)
    if err != nil {
        panic("configuration error, " + err.Error())
    }

    s3Client := s3.NewFromConfig(cfg)

    // Create an HTTP client.
    httpClient := &http.Client{
        Transport: http.DefaultTransport,
```

```

    }

    input := &s3.ListBucketsInput{}
    result, err := s3Client.ListBuckets(ctx, input)
    if err != nil {
        fmt.Printf("Got an error retrieving buckets, %v", err)
    }

    fmt.Println("Buckets:")
    for _, bucket := range result.Buckets {
        fmt.Println(*bucket.Name + ": " + bucket.CreationDate.Format("2006-01-02
15:04:05 Monday"))
    }
    fmt.Println("End Buckets.")

    req, err := http.NewRequestWithContext(ctx, http.MethodGet, "https://
api.github.com/repos/aws-observability/aws-otel-go/releases/latest", nil)
    if err != nil {
        fmt.Printf("failed to create http request, %v\n", err)
    }
    res, err := httpClient.Do(req)
    if err != nil {
        fmt.Printf("failed to make http request, %v\n", err)
    }
    defer func(Body io.ReadCloser) {
        err := Body.Close()
        if err != nil {
            fmt.Printf("failed to close http response body, %v\n", err)
        }
    }(res.Body)

    var data map[string]interface{}
    err = json.NewDecoder(res.Body).Decode(&data)
    if err != nil {
        fmt.Printf("failed to read http response body, %v\n", err)
    }
    fmt.Printf("Latest ADOT Go Release is '%s'\n", data["name"])

    return events.APIGatewayProxyResponse{
        StatusCode: http.StatusOK,
        Body:       os.Getenv("_X_AMZN_TRACE_ID"),
    }, nil
}

```

```
func main() {
    lambda.Start(lambdaHandler)
}
```

Pour instrumenter manuellement votre gestionnaire Lambda et le client Amazon S3, procédez comme suit :

1. Dans `main()`, instanciez un `TracerProvider` (`tp`) et enregistrez-le en tant que fournisseur de traceurs global. `TracerProvider` Il est recommandé de le configurer avec :
 - a. Processeur Span simple avec un exportateur de span UDP X-Ray pour envoyer des traces au point de terminaison UDP X-Ray de Lambda
 - b. Ressource avec `service.name` défini sur le nom de la fonction Lambda
2. Modifiez l'utilisation `lambda.Start(lambdaHandler)` en `otellambda.Start(otellambda.InstrumentHandler(lambdaHandler, xrayconfig.WithRecommendedOptions(tp)...))`.
3. Instrumentez le client Amazon S3 à l'aide de l'instrumentation du OpenTelemetry AWS SDK en ajoutant un `OpenTelemetry` intergiciel à la configuration du client `aws-sdk-go-v2` Amazon S3.
4. Instrumentez le client HTTP en utilisant OpenTelemetry la `otelhttp.NewTransport` méthode pour encapsuler `lehttp.DefaultTransport`.

Le code suivant est un exemple de ce à quoi ressemblera la fonction Lambda après les modifications. Vous pouvez créer manuellement des travées personnalisées supplémentaires en plus des travées fournies automatiquement.

```
package main

import (
    "context"
    "encoding/json"
    "fmt"
    "io"
    "net/http"
    "os"

    "github.com/aws-observability/aws-otel-go/exporters/xrayudp"
    "github.com/aws/aws-lambda-go/events"
    "github.com/aws/aws-lambda-go/lambda"
    awsconfig "github.com/aws/aws-sdk-go-v2/config"
)
```

```

"github.com/aws/aws-sdk-go-v2/service/s3"

    lambdadetector "go.opentelemetry.io/contrib/detectors/aws/lambda"
    "go.opentelemetry.io/contrib/instrumentation/github.com/aws/aws-lambda-go/
otelambda"
    "go.opentelemetry.io/contrib/instrumentation/github.com/aws/aws-lambda-go/
otelambda/xrayconfig"
    "go.opentelemetry.io/contrib/instrumentation/github.com/aws/aws-sdk-go-v2/
otelaws"
    "go.opentelemetry.io/contrib/instrumentation/net/http/otelhttp"
    "go.opentelemetry.io/contrib/propagators/aws/xray"
    "go.opentelemetry.io/otel"
    "go.opentelemetry.io/otel/attribute"
    "go.opentelemetry.io/otel/sdk/resource"
    "go.opentelemetry.io/otel/sdk/trace"
    semconv "go.opentelemetry.io/otel/semconv/v1.26.0"
)

func lambdaHandler(ctx context.Context) (interface{}, error) {
    // Initialize AWS config.
    cfg, err := awsconfig.LoadDefaultConfig(ctx)
    if err != nil {
        panic("configuration error, " + err.Error())
    }

    // Instrument all AWS clients.
    otelaws.AppendMiddlewares(&cfg.APIOptions)
    // Create an instrumented S3 client from the config.
    s3Client := s3.NewFromConfig(cfg)

    // Create an instrumented HTTP client.
    httpClient := &http.Client{
        Transport: otelhttp.NewTransport(
            http.DefaultTransport,
        ),
    }

    // return func(ctx context.Context) (interface{}, error) {
    input := &s3.ListBucketsInput{}
    result, err := s3Client.ListBuckets(ctx, input)
    if err != nil {
        fmt.Printf("Got an error retrieving buckets, %v", err)
    }
}

```

```

    fmt.Println("Buckets:")
    for _, bucket := range result.Buckets {
        fmt.Println(*bucket.Name + ": " + bucket.CreationDate.Format("2006-01-02
15:04:05 Monday"))
    }
    fmt.Println("End Buckets.")

    req, err := http.NewRequestWithContext(ctx, http.MethodGet, "https://
api.github.com/repos/aws-observability/aws-otel-go/releases/latest", nil)
    if err != nil {
        fmt.Printf("failed to create http request, %v\n", err)
    }
    res, err := httpClient.Do(req)
    if err != nil {
        fmt.Printf("failed to make http request, %v\n", err)
    }
    defer func(Body io.ReadCloser) {
        err := Body.Close()
        if err != nil {
            fmt.Printf("failed to close http response body, %v\n", err)
        }
    }(res.Body)

    var data map[string]interface{}
    err = json.NewDecoder(res.Body).Decode(&data)
    if err != nil {
        fmt.Printf("failed to read http response body, %v\n", err)
    }
    fmt.Printf("Latest ADOT Go Release is '%s'\n", data["name"])

    return events.APIGatewayProxyResponse{
        StatusCode: http.StatusOK,
        Body:       os.Getenv("_X_AMZN_TRACE_ID"),
    }, nil
}

func main() {
    ctx := context.Background()
    detector := lambdadetector.NewResourceDetector()
    lambdaResource, err := detector.Detect(context.Background())
    if err != nil {
        fmt.Printf("failed to detect lambda resources: %v\n", err)
    }
}

```

```
var attributes = []attribute.KeyValue{
    attribute.KeyValue{Key: semconv.ServiceNameKey, Value:
attribute.StringValue(os.Getenv("AWS_LAMBDA_FUNCTION_NAME"))},
}
customResource := resource.NewWithAttributes(semconv.SchemaURL, attributes...)
mergedResource, _ := resource.Merge(lambdaResource, customResource)

xrayUdpExporter, _ := xrayudp.NewSpanExporter(ctx)
tp := trace.NewTracerProvider(
    trace.WithSpanProcessor(trace.NewSimpleSpanProcessor(xrayUdpExporter)),
    trace.WithResource(mergedResource),
)

defer func(ctx context.Context) {
    err := tp.Shutdown(ctx)
    if err != nil {
        fmt.Printf("error shutting down tracer provider: %v", err)
    }
}(ctx)

otel.SetTracerProvider(tp)
otel.SetTextMapPropagator(xray.Propagator{})

lambda.Start(otelambda.InstrumentHandler(lambdaHandler,
xrayconfig.WithRecommendedOptions(tp)...))
}
```

Lorsque vous appelez Lambda, vous verrez la trace suivante dans Trace Map CloudWatch la console :



Migrer vers OpenTelemetry Node.js

Cette section explique comment migrer vos applications Node.js du SDK X-Ray vers OpenTelemetry. Il couvre à la fois les approches d'instrumentation automatique et manuelle et fournit des exemples spécifiques pour des cas d'utilisation courants.

Le SDK X-Ray Node.js vous permet d'instrumenter manuellement vos applications Node.js pour le traçage. Cette section fournit des exemples de code pour la migration de X-Ray vers l'OpenTelemetry instrumentation.

Sections

- [Solutions d'instrumentation automatique à code zéro](#)
- [Solutions d'instrumentation manuelle](#)
- [Suivi des demandes entrantes](#)
- [AWS Instrumentation du SDK JavaScript V3](#)
- [Instrumentation des appels HTTP sortants](#)
- [Support d'instrumentation pour d'autres bibliothèques](#)
- [Création manuelle de données de trace](#)
- [Instrumentation Lambda](#)

Solutions d'instrumentation automatique à code zéro

Pour suivre les demandes avec le SDK X-Ray pour Node.js, vous devez modifier le code de votre application. Avec OpenTelemetry, vous pouvez utiliser des solutions d'instrumentation automatique sans code pour suivre les demandes.

Instrumentation automatique à code zéro avec instrumentations automatiques OpenTelemetry basées sur la base.

1. Utilisation de la AWS distribution pour l'instrumentation automatique OpenTelemetry (ADOT) pour Node.js — Pour l'instrumentation automatique pour l'application Node.js, voir [Tracing and Metrics with the AWS Distro](#) for Auto-Instrumentation. OpenTelemetry JavaScript

(Facultatif) Vous pouvez également activer les signaux CloudWatch d'application lorsque vous instrumentez automatiquement vos applications à l' AWS aide de l' JavaScript instrumentation automatique ADOT afin de surveiller l'état actuel des applications et de suivre les performances des applications à long terme par rapport à vos objectifs commerciaux. Application Signals vous offre une vue unifiée et orientée application de vos services, dépendances et applications, et vous aide à surveiller et à trier l'état des applications. Pour plus d'informations, veuillez consulter la rubrique [Applications Signals](#).

2. Utilisation de l'instrumentation automatique OpenTelemetry JavaScript à code zéro — Pour l'instrumentation automatique avec le OpenTelemetry JavaScript, voir instrumentation [JavaScript à code zéro](#).

Solutions d'instrumentation manuelle

Tracing setup with X-Ray SDK

Lorsque le SDK X-Ray pour Node.js était utilisé, le `aws-xray-sdk` package devait configurer le SDK X-Ray avec des plug-ins de service ou des règles d'échantillonnage locales avant d'utiliser le SDK pour instrumenter votre code.

```
var AWSXRay = require('aws-xray-sdk');

AWSXRay.config([AWSXRay.plugins.EC2Plugin, AWSXRay.plugins.ElasticBeanstalkPlugin]);
AWSXRay.middleware.setSamplingRules(<path to file>);
```

Tracing setup with OpenTelemetry SDK

Note

AWS X-Ray Remote Sampling n'est actuellement pas disponible pour être configuré pour OpenTelemetry JS. Cependant, la prise en charge de l'échantillonnage à distance X-Ray est actuellement disponible via l'ADOT Auto-Instrumentation pour Node.js.

Pour l'exemple de code ci-dessous, vous aurez besoin des dépendances suivantes :

```
npm install --save \  
  @opentelemetry/api \  
  @opentelemetry/sdk-node \  
  @opentelemetry/exporter-trace-otlp-proto \  
  @opentelemetry/propagator-aws-xray \  
  @opentelemetry/resource-detector-aws
```

Vous devez installer et configurer le OpenTelemetry SDK avant d'exécuter le code de votre application. Cela peut être fait en utilisant l'indicateur [—require](#). Créez un fichier nommé instrumentation.js, qui contiendra la configuration et l'installation de votre OpenTelemetry instrumentation.

Il est recommandé de configurer les composants suivants :

- OTLPTraceExportateur — Obligatoire pour exporter des traces vers l' CloudWatch OpenTelemetry agent/le collecteur
- AWSXRayPropagator — Nécessaire pour propager le contexte de trace aux AWS services intégrés à X-Ray
- Détecteurs de ressources (par exemple, Amazon EC2 Resource Detector) : pour détecter les métadonnées de l'hôte exécutant votre application

```
/*instrumentation.js*/  
// Require dependencies  
const { NodeSDK } = require('@opentelemetry/sdk-node');
```

```
const { OTLPTraceExporter } = require('@opentelemetry/exporter-trace-otlp-proto');
const { AWSXRayPropagator } = require("@opentelemetry/propagator-aws-xray");
const { detectResources } = require('@opentelemetry/resources');
const { awsEc2Detector } = require('@opentelemetry/resource-detector-aws');

const resource = detectResources({
  detectors: [awsEc2Detector],
});

const _traceExporter = new OTLPTraceExporter({
  url: 'http://localhost:4318/v1/traces'
});

const sdk = new NodeSDK({
  resource: resource,
  textMapPropagator: new AWSXRayPropagator(),
  traceExporter: _traceExporter
});

sdk.start();
```

Ensuite, vous pouvez exécuter votre application avec votre OpenTelemetry configuration comme suit :

```
node --require ./instrumentation.js app.js
```

Vous pouvez utiliser les instruments de bibliothèque du OpenTelemetry SDK pour créer automatiquement des étendues pour des bibliothèques telles que le SDK. AWS Leur activation créera automatiquement des intervalles pour les modules tels que le AWS SDK pour JavaScript la version 3. OpenTelemetry offre la possibilité d'activer toutes les instrumentations de bibliothèque ou de spécifier les instrumentations de bibliothèque à activer.

Pour activer toutes les instrumentations, installez le `@opentelemetry/auto-instrumentations-node` package :

```
npm install @opentelemetry/auto-instrumentations-node
```

Ensuite, mettez à jour la configuration pour activer toutes les instrumentations de bibliothèque, comme indiqué ci-dessous.

```
const { getNodeAutoInstrumentations } = require('@opentelemetry/auto-instrumentations-node');

...

const sdk = new NodeSDK({
  resource: resource,
  instrumentations: [getNodeAutoInstrumentations()],
  textMapPropagator: new AWSXRayPropagator(),
  traceExporter: _traceExporter
});
```

Tracing setup with ADOT auto-instrumentation for Node.js

Vous pouvez utiliser l'instrumentation automatique ADOT pour Node.js afin de configurer automatiquement vos OpenTelemetry applications Node.js. En utilisant ADOT Auto-Instrumentation, vous n'avez pas besoin de modifier manuellement le code pour suivre les demandes entrantes ou pour suivre les bibliothèques telles que le AWS SDK ou les clients HTTP. Pour plus d'informations, consultez la section [Traçage et métriques avec le AWS distributeur pour l'instrumentation OpenTelemetry JavaScript automatique](#).

L'instrumentation automatique ADOT pour Node.js prend en charge :

- Échantillonnage à distance par rayons X via une variable d'environnement — `export OTEL_TRACES_SAMPLER=xray`
- Propagation du contexte X-Ray Trace (activée par défaut)
- Détection des ressources (la détection des ressources pour les environnements Amazon EC2, Amazon ECS et Amazon EKS est activée par défaut)
- Instrumentations de bibliothèque automatiques pour toutes les OpenTelemetry instrumentations prises en charge, qui peuvent être sélectionnées de disabled/enabled manière sélective par le biais de variables d'environnement `OTEL_NODE_ENABLED_INSTRUMENTATION` et `OTEL_NODE_DISABLED_INSTRUMENTATIONS`
- Création manuelle de Spans

Suivi des demandes entrantes

With X-Ray SDK

Express.js

Avec le SDK X-Ray pour suivre les requêtes HTTP entrantes reçues par les applications Express.js, les deux `AWSXRay.express.openSegment(<name>)` intergiciels `AWSXRay.express.closeSegment()` devaient encapsuler toutes les routes définies afin de les tracer.

```
app.use(xrayExpress.openSegment('defaultName'));  
  
...  
  
app.use(xrayExpress.closeSegment());
```

Restifier

Pour suivre les requêtes HTTP entrantes reçues par Restify les applications, le middleware du SDK X-Ray a été utilisé en exécutant enable depuis le `aws-xray-sdk-restify` module sur le serveur 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');
```

With OpenTelemetry SDK

Express.js

Le suivi des demandes entrantes Express.js est assuré par l'[instrumentation OpenTelemetry HTTP](#) et l'[instrumentation OpenTelemetry express](#). Installez les dépendances suivantes avec npm :

```
npm install --save @opentelemetry/instrumentation-http @opentelemetry/instrumentation-express
```

Mettez à jour la configuration du OpenTelemetry SDK pour activer l'instrumentation pour le module express :

```
const { HttpInstrumentation } = require('@opentelemetry/instrumentation-http');
const { ExpressInstrumentation } = require('@opentelemetry/instrumentation-express');
...

const sdk = new NodeSDK({
  ...

  instrumentations: [
    ...
    // Express instrumentation requires HTTP instrumentation
    new HttpInstrumentation(),
    new ExpressInstrumentation(),
  ],
});
```

Restifier

Pour les applications Restify, vous aurez besoin de l'instrumentation [OpenTelemetry Restify](#). Installez la dépendance suivante :

```
npm install --save @opentelemetry/instrumentation-restify
```

Mettez à jour la configuration du OpenTelemetry SDK pour activer l'instrumentation pour le module restify :

```
const { RestifyInstrumentation } = require('@opentelemetry/instrumentation-restify');
```

```
...

const sdk = new NodeSDK({
  ...

  instrumentations: [
    ...
    new RestifyInstrumentation(),
  ],
});
```

AWS Instrumentation du SDK JavaScript V3

With X-Ray SDK

Pour instrumenter les AWS demandes sortantes depuis le AWS SDK, vous avez instrumenté les clients comme dans l'exemple suivant :

```
import { S3, PutObjectCommand } from '@aws-sdk/client-s3';

const s3 = AWSXRay.captureAWSSv3Client(new S3({}));

await s3.send(new PutObjectCommand({
  Bucket: bucketName,
  Key: keyName,
  Body: 'Hello!',
}));
```

With OpenTelemetry SDK

La prise en charge du suivi des appels du AWS SDK en aval vers DynamoDB, Amazon S3 et d'autres est fournie par OpenTelemetry AWS l'instrumentation du SDK. Installez la dépendance suivante avec npm :

```
npm install --save @opentelemetry/instrumentation-aws-sdk
```

Mettez à jour la configuration du OpenTelemetry SDK avec l'instrumentation du AWS SDK.

```
import { AwsInstrumentation } from '@opentelemetry/instrumentation-aws-sdk';
...

const sdk = new NodeSDK({
  ...

  instrumentations: [
    ...
    new AwsInstrumentation()
  ],
});
```

Instrumentation des appels HTTP sortants

With X-Ray SDK

Pour instrumenter les requêtes HTTP sortantes avec X-Ray, il était nécessaire d'instrumenter les clients. Par exemple, voir ci-dessous.

Clients HTTP individuels

```
var AWSXRay = require('aws-xray-sdk');
var http = AWSXRay.captureHTTPs(require('http'));
```

Tous les clients HTTP (globaux)

```
var AWSXRay = require('aws-xray-sdk');
AWSXRay.captureHTTPsGlobal(require('http'));
var http = require('http');
```

With OpenTelemetry SDK

La prise en charge du suivi pour les clients HTTP Node.js est fournie par l'instrumentation OpenTelemetry HTTP. Installez la dépendance suivante avec npm :


```
npm install --save @opentelemetry/instrumentation-http
```

Mettez à jour la configuration du OpenTelemetry SDK comme suit :

```
const { HttpInstrumentation } = require('@opentelemetry/instrumentation-http');
...

const sdk = new NodeSDK({
  ...

  instrumentations: [
    ...
    new HttpInstrumentation(),
  ],
});
```

Support d'instrumentation pour d'autres bibliothèques

Vous trouverez la liste complète des instruments de bibliothèque pris en charge OpenTelemetry JavaScript sous Instrumentations [prises en charge](#).

Vous pouvez également effectuer une recherche OpenTelemetry dans le registre pour savoir si l'instrumentation est OpenTelemetry compatible avec votre bibliothèque sous [Registre](#).

Création manuelle de données de trace

With X-Ray SDK

À l'aide de X-Ray, le code `aws-xray-sdk` du package était nécessaire pour créer manuellement des segments et leurs sous-segments secondaires afin de suivre votre application.

```
var AWSXRay = require('aws-xray-sdk');

AWSXRay.enableManualMode();
```

```
var segment = new AWSXRay.Segment('myApplication');

captureFunc('1', function(subsegment1) {
  captureFunc('2', function(subsegment2) {

    }, subsegment1);
  }, segment);

segment.close();
segment.flush();
```

With OpenTelemetry SDK

Vous pouvez créer et utiliser des intervalles personnalisés pour surveiller les performances des activités internes qui ne sont pas capturées par les bibliothèques d'instrumentation. Notez que seules les plages de type Server sont converties en segments X-Ray, toutes les autres plages sont converties en sous-segments de X-Ray. Pour plus d'informations, consultez [Segments](#).

Vous aurez besoin d'une instance Tracer après avoir configuré le OpenTelemetry SDK dans Tracing Setup pour créer des Spans. Vous pouvez créer autant d'instances de Tracer que nécessaire, mais il est courant d'avoir un seul Tracer pour l'ensemble d'une application.

```
const { trace, SpanKind } = require('@opentelemetry/api');

// Get a tracer instance
const tracer = trace.getTracer('your-tracer-name');

...

// This span will appear as a segment in X-Ray
tracer.startActiveSpan('server', { kind: SpanKind.SERVER }, span => {
  // Do work here

  // This span will appear as a subsegment in X-Ray
  tracer.startActiveSpan('operation2', { kind: SpanKind.INTERNAL }, innerSpan => {
    // Do more work here

    innerSpan.end();
  });
  span.end();
});
```

Ajouter des annotations et des métadonnées aux traces avec OpenTelemetry le SDK

Vous pouvez également ajouter des paires clé-valeur personnalisées en tant qu'attributs dans vos spans. Notez que par défaut, tous ces attributs d'étendue seront convertis en métadonnées dans les données brutes de X-Ray. Pour vous assurer qu'un attribut est converti en annotation et non en métadonnées, ajoutez la clé de l'attribut à la liste de l'`aws.xray.annotationsattribut`. Pour plus d'informations, consultez [Enable The Customized X-Ray Annotations](#).

```
tracer.startActiveSpan('server', { kind: SpanKind.SERVER }, span => {
  span.setAttribute('metadataKey', 'metadataValue');
  span.setAttribute('annotationKey', 'annotationValue');

  // The following ensures that "annotationKey: annotationValue" is an annotation
  in X-Ray raw data.
  span.setAttribute('aws.xray.annotations', ['annotationKey']);

  // Do work here

  span.end();
});
```

Instrumentation Lambda

With X-Ray SDK

Après avoir activé Active Tracing pour votre fonction Lambda, le SDK X-Ray était requis sans configuration supplémentaire. Lambda crée un segment représentant l'invocation du gestionnaire Lambda, et vous avez créé des sous-segments ou des bibliothèques d'instruments à l'aide du SDK X-Ray sans aucune configuration supplémentaire.

With OpenTelemetry SDK

Vous pouvez automatiquement instrumenter votre Lambda avec des couches AWS Lambda vendues. Il existe deux solutions :

- Couche Lambda (recommandée) des signaux d' CloudWatch application

 Note

Les signaux d' CloudWatch application sont activés par défaut sur cette couche Lambda, ce qui permet de surveiller les performances et l'état de santé de votre application Lambda en collectant à la fois des métriques et des traces. Si vous souhaitez uniquement du traçage, vous devez définir la variable d'environnement Lambda. `OTEL_AWS_APPLICATION_SIGNALS_ENABLED=false` Pour plus d'informations, consultez [Activer vos applications sur Lambda](#).

- AWS couche Lambda gérée pour ADOT JS. Pour plus d'informations, consultez [AWS Distro for OpenTelemetry Lambda Support JavaScript](#) for.

Création manuelle de spans avec l'instrumentation Lambda

Bien que la couche JavaScript Lambda ADOT fournisse une instrumentation automatique pour votre fonction Lambda, vous pourriez avoir besoin d'effectuer une instrumentation manuelle dans votre Lambda, par exemple, pour fournir des données personnalisées ou pour instrumenter du code au sein de la fonction Lambda elle-même qui n'est pas couvert par les instrumentations de bibliothèque.

Pour effectuer une instrumentation manuelle parallèlement à l'instrumentation automatique, vous devez ajouter `@opentelemetry/api` une dépendance. Il est recommandé que la version de cette dépendance soit la même version de la même dépendance que celle utilisée par le JavaScript SDK ADOT. Vous pouvez utiliser l' OpenTelemetry API pour créer manuellement des intervalles dans votre fonction Lambda.

Pour ajouter la `@opentelemetry/api` dépendance à l'aide de NPM :

```
npm install @opentelemetry/api
```

Migrer vers OpenTelemetry .NET

Lorsque vous utilisez X-Ray Tracing dans vos applications .NET, le SDK X-Ray .NET associé à des efforts manuels est utilisé pour l'instrumentation.

Cette section fournit des exemples de code dans la [Solutions d'instrumentation manuelle avec le SDK](#) section relative à la migration de la solution d'instrumentation manuelle X-Ray vers des solutions OpenTelemetry d'instrumentation manuelle pour .NET. Vous pouvez également passer de l'instrumentation manuelle X-Ray aux solutions d'instrumentation OpenTelemetry automatique pour les applications .NET d'instrumentation sans avoir à modifier le code source de l'application dans la [Solutions d'instrumentation automatique à code zéro](#) section.

Sections

- [Solutions d'instrumentation automatique à code zéro](#)
- [Solutions d'instrumentation manuelle avec le SDK](#)
- [Création manuelle de données de trace](#)
- [Suivi des demandes entrantes \(ASP.NET et instrumentation de base ASP.NET\)](#)
- [AWS Instrumentation du SDK](#)
- [Instrumentation des appels HTTP sortants](#)
- [Support d'instrumentation pour d'autres bibliothèques](#)
- [Instrumentation Lambda](#)

Solutions d'instrumentation automatique à code zéro

OpenTelemetry fournit des solutions d'instrumentation automatique sans code. Ces solutions permettent de suivre les demandes sans qu'il soit nécessaire de modifier le code de votre application.

OpenTelemetry options d'instrumentation automatique basées sur des options

1. Utilisation de la AWS distribution pour l'instrumentation automatique OpenTelemetry (ADOT) pour .NET — Pour instrumenter automatiquement les applications .NET, reportez-vous à la section [Suivi et mesures avec la AWS distribution pour l'instrumentation automatique .NET](#). OpenTelemetry

(Facultatif) Activez les signaux d' CloudWatch application lorsque vous instrumentez automatiquement vos applications AWS avec l'instrumentation automatique ADOT .NET pour :

- Surveiller l'état actuel des applications
- Suivez les performances des applications à long terme par rapport aux objectifs commerciaux
- Bénéficiez d'une vue unifiée et centrée sur les applications de vos applications, services et dépendances

- Surveillez et trie l'état de santé des applications

Pour plus d'informations, veuillez consulter la rubrique [Applications Signals](#).

2. Utilisation de l'instrumentation automatique sans code OpenTelemetry .Net : pour instrumenter automatiquement avec OpenTelemetry .Net, voir [Tracing and Metrics with the AWS Distro pour OpenTelemetry .NET](#) Auto-Instrumentation.

Solutions d'instrumentation manuelle avec le SDK

Tracing configuration with X-Ray SDK

Pour les applications Web .NET, le SDK X-Ray est configuré dans la section AppSettings du Web.config fichier.

Exemple Web.config

```
<configuration>
  <appSettings>
    <add key="AWSXRayPlugins" value="EC2Plugin"/>
  </appSettings>
</configuration>
```

Pour .NET Core, un fichier nommé appsettings.json avec une clé de niveau supérieur XRay est utilisé, puis un objet de configuration est créé pour initialiser l'enregistreur X-Ray.

Exemple pour .NET appsettings.json

```
{
  "XRay": {
    "AWSXRayPlugins": "EC2Plugin"
  }
}
```

Exemple pour .NET Core Program.cs — Configuration de l'enregistreur

```
using Amazon.XRay.Recorder.Core;
...
AWSXRayRecorder.InitializeInstance(configuration);
```

Tracing configuration with OpenTelemetry SDK

Ajoutez les dépendances suivantes :

```
dotnet add package OpenTelemetry
dotnet add package OpenTelemetry.Contrib.Extensions.AWSXRay
dotnet add package OpenTelemetry.Sampler.AWS --prerelease
dotnet add package OpenTelemetry.Resources.AWS
dotnet add package OpenTelemetry.Exporter.OpenTelemetryProtocol
dotnet add package OpenTelemetry.Extensions.Hosting
dotnet add package OpenTelemetry.Instrumentation.AspNetCore
```

Pour votre application .NET, configurez le OpenTelemetry SDK en configurant le Global TracerProvider. L'exemple de configuration suivant active également l'instrumentation pour ASP.NET Core. Pour instrumenter ASP.NET, voir [Suivi des demandes entrantes \(ASP.NET et instrumentation de base ASP.NET\)](#). Pour OpenTelemetry une utilisation avec d'autres frameworks, voir [Registry](#) pour plus de bibliothèques pour les frameworks pris en charge.

Il est recommandé de configurer les composants suivants :

- An OTLP Exporter— Nécessaire pour exporter les traces vers l' CloudWatch OpenTelemetry agent/le collecteur
- Un propagateur de AWS rayons X : nécessaire pour propager le contexte de trace aux [AWS services intégrés à X-Ray](#)
- Un échantillonneur à distance AWS X-Ray : obligatoire si vous devez prélever des [échantillons à l'aide des règles d'échantillonnage X-Ray](#)
- Resource Detectors(par exemple, Amazon EC2 Resource Detector) : pour détecter les métadonnées de l'hôte exécutant votre application

```
using OpenTelemetry;
```

```

using OpenTelemetry.Contrib.Extensions.AWSXRay.Trace;
using OpenTelemetry.Sampler.AWS;
using OpenTelemetry.Trace;
using OpenTelemetry.Resources;

var builder = WebApplication.CreateBuilder(args);

var serviceName = "MyServiceName";
var serviceVersion = "1.0.0";

var resourceBuilder = ResourceBuilder
    .CreateDefault()
    .AddService(serviceName: serviceName)
    .AddAWSEC2Detector();

builder.Services.AddOpenTelemetry()
    .ConfigureResource(resource => resource
        .AddAWSEC2Detector()
        .AddService(
            serviceName: serviceName,
            serviceVersion: serviceVersion))
    .WithTracing(tracing => tracing
        .AddSource(serviceName)
        .AddAspNetCoreInstrumentation()
        .AddOtlpExporter()
        .SetSampler(AWSXRayRemoteSampler.Builder(resourceBuilder.Build())
            .SetEndpoint("http://localhost:2000")
            .Build()));

Sdk.SetDefaultTextMapPropagator(new AWSXRayPropagator()); // configure X-Ray
propagator

```

OpenTelemetry Pour l'utiliser pour une application console, ajoutez la OpenTelemetry configuration suivante au démarrage de votre programme.

```

using OpenTelemetry;
using OpenTelemetry.Contrib.Extensions.AWSXRay.Trace;
using OpenTelemetry.Trace;
using OpenTelemetry.Resources;

var serviceName = "MyServiceName";

```



```

var resourceBuilder = ResourceBuilder
    .CreateDefault()
    .AddService(serviceName: serviceName)
    .AddAWSEC2Detector();

var tracerProvider = Sdk.CreateTracerProviderBuilder()
    .AddSource(serviceName)
    .ConfigureResource(resource =>
        resource
            .AddAWSEC2Detector()
            .AddService(
                serviceName: serviceName,
                serviceVersion: serviceVersion
            )
    )
    .AddOtlpExporter() // default address localhost:4317
    .SetSampler(new TraceIdRatioBasedSampler(1.00))
    .Build();

Sdk.SetDefaultTextMapPropagator(new AWSXRayPropagator()); // configure X-Ray
propagator

```

Création manuelle de données de trace

With X-Ray SDK

Avec le SDK X-Ray, les `BeginSubsegment` méthodes `BeginSegment` et étaient nécessaires pour créer manuellement des segments et des sous-segments X-Ray.

```

using Amazon.XRay.Recorder.Core;

AWSXRayRecorder.Instance.BeginSegment("segment name"); // generates `TraceId` for
you
try
{
    // Do something here
    // can create custom subsegments
    AWSXRayRecorder.Instance.BeginSubsegment("subsegment name");
    try

```

```
{
    DoSomething();
}
catch (Exception e)
{
    AWSXRayRecorder.Instance.AddException(e);
}
finally
{
    AWSXRayRecorder.Instance.EndSubsegment();
}
}
catch (Exception e)
{
    AWSXRayRecorder.Instance.AddException(e);
}
finally
{
    AWSXRayRecorder.Instance.EndSegment();
}
```

With OpenTelemetry SDK

Dans .NET, vous pouvez utiliser l'API d'activité pour créer des intervalles personnalisés afin de surveiller les performances des activités internes qui ne sont pas capturées par les bibliothèques d'instrumentation. Notez que seules les pages de type Server sont converties en segments X-Ray, toutes les autres pages sont converties en sous-segments de X-Ray.

Vous pouvez créer autant d'`ActivitySource` instances que nécessaire, mais il est recommandé de n'en avoir qu'une pour l'ensemble d'une application/d' un service.

```
using System.Diagnostics;

ActivitySource activitySource = new ActivitySource("ActivitySourceName",
    "ActivitySourceVersion");

...
```

```
using (var activity = activitySource.StartActivity("ActivityName",
    ActivityKind.Server)) // this will be translated to a X-Ray Segment
{
    // Do something here

    using (var internalActivity = activitySource.StartActivity("ActivityName",
        ActivityKind.Internal)) // this will be translated to an X-Ray Subsegment
    {
        // Do something here
    }
}
```

Ajouter des annotations et des métadonnées aux traces avec OpenTelemetry le SDK

Vous pouvez également ajouter des paires clé-valeur personnalisées en tant qu'attributs à vos spans en utilisant la `SetTag` méthode appliquée à une activité. Notez que par défaut, tous les attributs de span seront convertis en métadonnées dans les données brutes de X-Ray. Pour garantir qu'un attribut est converti en annotation et non en métadonnées, vous pouvez ajouter la clé de cet attribut à la liste des `aws.xray.annotations` attributs.

```
using (var activity = activitySource.StartActivity("ActivityName",
    ActivityKind.Server)) // this will be translated to a X-Ray Segment
{
    activity.SetTag("metadataKey", "metadataValue");
    activity.SetTag("annotationKey", "annotationValue");
    string[] annotationKeys = {"annotationKey"};
    activity.SetTag("aws.xray.annotations", annotationKeys);

    // Do something here

    using (var internalActivity = activitySource.StartActivity("ActivityName",
        ActivityKind.Internal)) // this will be translated to an X-Ray Subsegment
    {
        // Do something here
    }
}
```

Avec instrumentation OpenTelemetry automatique

Si vous utilisez une solution d'instrumentation OpenTelemetry automatique pour .NET et si vous devez effectuer une instrumentation manuelle dans votre application, par exemple, pour coder des instruments dans l'application elle-même pour les sections qui ne sont couvertes par aucune bibliothèque d'instrumentation automatique.

Comme il ne peut y avoir qu'une seule instrumentation globale `TracerProvider`, l'instrumentation manuelle ne doit pas instancier la sienne `TracerProvider` si elle est utilisée conjointement avec l'instrumentation automatique. Lorsqu'il `TracerProvider` est utilisé, le traçage manuel personnalisé fonctionne de la même manière lors de l'utilisation d'une instrumentation automatique ou manuelle via le OpenTelemetry SDK.

Suivi des demandes entrantes (ASP.NET et instrumentation de base ASP.NET)

With X-Ray SDK

Pour instrumenter les requêtes traitées par l'application ASP.NET, reportez-vous à la section <https://docs.aws.amazon.com/xray/latest/devguide/xray-sdk-dotnet-messagehandler.html> pour savoir comment appeler `RegisterXRay` la `Init` méthode de votre `global.asax` fichier.

```
AWSXRayASPNET.RegisterXRay(this, "MyApp");
```

Pour instrumenter les requêtes traitées par votre application principale ASP.NET, la `UseXRay` méthode est appelée avant tout autre intergiciel dans la `Configure` méthode de votre classe `Startup`.

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

With OpenTelemetry SDK

OpenTelemetry fournit également des bibliothèques d'instrumentation pour collecter des traces pour les requêtes Web entrantes pour ASP.NET et ASP.NET core. [La section suivante répertorie les étapes nécessaires pour ajouter et activer ces instrumentations de bibliothèque pour votre](#)

[OpenTelemetry configuration, notamment comment ajouter l'instrumentation principale ASP.NET ou ASP.NET lors de la création du fournisseur de traceurs.](#)

Pour plus d'informations sur la façon d'activer OpenTelemetry .Instrumentation. AspNet, voir [Étapes pour activer OpenTelemetry .Instrumentation. AspNet](#) et pour obtenir des informations sur la façon d'activer OpenTelemetry .Instrumentation. AspNetCore, voir [Étapes pour activer OpenTelemetry .Instrumentation. AspNetCore](#).

AWS Instrumentation du SDK

With X-Ray SDK

Installez tous les clients du AWS SDK en appelant `RegisterXRayForAllServices()`.

```
using Amazon.XRay.Recorder.Handlers.AwsSdk;
AWSSDKHandler.RegisterXRayForAllServices(); //place this before any instantiation of
AmazonServiceClient
AmazonDynamoDBClient client = new AmazonDynamoDBClient(RegionEndpoint.USWest2); //
AmazonDynamoDBClient is automatically registered with X-Ray
```

Utilisez l'une des méthodes suivantes pour l'instrumentation spécifique du client de AWS service.

```
AWSSDKHandler.RegisterXRay<IAmazonDynamoDB>(); // Registers specific type of
AmazonServiceClient : All instances of IAmazonDynamoDB created after this line are
registered
AWSSDKHandler.RegisterXRayManifest(String path); // To configure custom AWS Service
Manifest file. This is optional, if you have followed "Configuration" section
```

With OpenTelemetry SDK

Pour l'exemple de code suivant, vous aurez besoin de la dépendance suivante :

```
dotnet add package OpenTelemetry.Instrumentation.AWS
```

Pour instrumenter le AWS SDK, mettez à jour la configuration du OpenTelemetry SDK dans laquelle le Global TracerProvider est configuré.

```
builder.Services.AddOpenTelemetry()  
    ...  
    .WithTracing(tracing => tracing  
        .AddAWSInstrumentation()  
        ...
```

Instrumentation des appels HTTP sortants

With X-Ray SDK

Le SDK .NET de X-Ray trace les appels HTTP sortants via les méthodes d'extension `GetResponseTraced()` ou `GetAsyncResponseTraced()` lors de l'utilisation `System.Net.HttpWebRequest`, ou en utilisant le `HttpClientXRayTracingHandler` gestionnaire lors de l'utilisation. `System.Net.Http.HttpClient`

With OpenTelemetry SDK

Pour l'exemple de code suivant, vous aurez besoin de la dépendance suivante :

```
dotnet add package OpenTelemetry.Instrumentation.Http
```

Pour instrumenter `System.Net.Http.HttpClient` et `System.Net.HttpWebRequest` mettre à jour la configuration du OpenTelemetry SDK où le Global TracerProvider est configuré.

```
builder.Services.AddOpenTelemetry()  
    ...  
    .WithTracing(tracing => tracing  
        .AddHttpClientInstrumentation()  
        ...
```

Support d'instrumentation pour d'autres bibliothèques

Vous pouvez rechercher et filtrer les bibliothèques d'instrumentation .NET OpenTelemetry dans le registre afin de déterminer si votre bibliothèque OpenTelemetry prend en charge l'instrumentation. Consultez le [registre](#) pour commencer la recherche.

Instrumentation Lambda

With X-Ray SDK

La procédure suivante était requise pour utiliser le SDK X-Ray avec Lambda :

1. Activez le suivi actif sur votre fonction Lambda
2. Le service Lambda crée un segment qui représente l'invocation de votre gestionnaire
3. Créez des sous-segments ou des bibliothèques d'instruments à l'aide du SDK X-Ray

With OpenTelemetry-based solutions

Vous pouvez automatiquement instrumenter votre Lambda avec des couches AWS Lambda vendues. Il existe deux solutions :

- Couche [lambda \(recommandée\) des signaux d'CloudWatch application](#)
- Pour de meilleures performances, vous pouvez envisager de générer des OpenTelemetry traces pour votre fonction Lambda. [OpenTelemetry Manual Instrumentation](#)

OpenTelemetry instrumentation manuelle pour AWS Lambda

Voici un exemple de code de fonction Lambda (sans instrumentation).

```
using System;
using System.Text;
using System.Threading.Tasks;
using Amazon.Lambda.Core;
using Amazon.S3;
using Amazon.S3.Model;

// Assembly attribute to enable Lambda function logging
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))
]
```

```
namespace ExampleLambda;

public class ListBucketsHandler
{
    private static readonly AmazonS3Client s3Client = new();

    // new Lambda function handler passed in
    public async Task<string> HandleRequest(object input, ILambdaContext context)
    {
        try
        {
            var DoListBucketsAsyncResponse = await DoListBucketsAsync();
            context.Logger.LogInformation($"Results:
{DoListBucketsAsyncResponse.Buckets}");

            context.Logger.LogInformation($"Successfully called ListBucketsAsync");
            return "Success!";
        }
        catch (Exception ex)
        {
            context.Logger.LogError($"Failed to call ListBucketsAsync: {ex.Message}");
            throw;
        }
    }

    private async Task<ListBucketsResponse> DoListBucketsAsync()
    {
        try
        {
            var putRequest = new ListBucketsRequest
            {
            };

            var response = await s3Client.ListBucketsAsync(putRequest);
            return response;
        }
        catch (AmazonS3Exception ex)
        {
            throw new Exception($"Failed to call ListBucketsAsync: {ex.Message}", ex);
        }
    }
}
```


Pour instrumenter manuellement votre gestionnaire Lambda et le client Amazon S3, procédez comme suit.

1. Instancier a TracerProvider — Il TracerProvider est recommandé de le configurer avec unXrayUdpSpanExporter, un échantillonneur ParentBased Always On et un Resource with service.name définis sur le nom de la fonction Lambda.
2. Instrumenter le client Amazon S3 avec l'instrumentation du OpenTemetry AWS SDK en appelant AddAWSInstrumentation() pour ajouter l'instrumentation du client du AWS SDK à TracerProvider
3. Créez une fonction wrapper avec la même signature que la fonction Lambda d'origine. Appelez AWSLambdaWrapper.Trace() l'API et transmettez TracerProvider la fonction Lambda d'origine et ses entrées en tant que paramètres. Définissez la fonction wrapper comme entrée du gestionnaire Lambda.

Pour l'exemple de code suivant, vous aurez besoin des dépendances suivantes :

```
dotnet add package OpenTelemetry.Instrumentation.AWSLambda
dotnet add package OpenTelemetry.Instrumentation.AWS
dotnet add package OpenTelemetry.Resources.AWS
dotnet add package AWS.Distro.OpenTelemetry.Exporter.Xray.Udp
```

Le code suivant illustre la fonction Lambda après les modifications requises. Vous pouvez créer des travées personnalisées supplémentaires pour compléter les travées fournies automatiquement.

```
using Amazon.Lambda.Core;
using Amazon.S3;
using Amazon.S3.Model;
using OpenTelemetry;
using OpenTelemetry.Instrumentation.AWSLambda;
using OpenTelemetry.Trace;
using AWS.Distro.OpenTelemetry.Exporter.Xray.Udp;
using OpenTelemetry.Resources;

// Assembly attribute to enable Lambda function logging
```

```
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))

namespace ExampleLambda;

public class ListBucketsHandler
{
    private static readonly AmazonS3Client s3Client = new();

    TracerProvider tracerProvider = Sdk.CreateTracerProviderBuilder()
        .AddAWSLambdaConfigurations()
        .AddProcessor(
            new SimpleActivityExportProcessor(
                // AWS_LAMBDA_FUNCTION_NAME Environment Variable will be defined in AWS
                Lambda Environment
                new
                XrayUdpExporter(ResourceBuilder.CreateDefault().AddService(Environment.GetEnvironmentVariable(
                    )
                )
                )
                .AddAWSInstrumentation()
                .SetSampler(new ParentBasedSampler(new AlwaysOnSampler()))
                .Build();

    // new Lambda function handler passed in
    public async Task<string> HandleRequest(object input, ILambdaContext context)
    => await AWSLambdaWrapper.Trace(tracerProvider, OriginalHandleRequest, input,
    context);

    public async Task<string> OriginalHandleRequest(object input, ILambdaContext
    context)
    {
        try
        {
            var DoListBucketsAsyncResponse = await DoListBucketsAsync();
            context.Logger.LogInformation($"Results:
            {DoListBucketsAsyncResponse.Buckets}");

            context.Logger.LogInformation($"Successfully called ListBucketsAsync");
            return "Success!";
        }
        catch (Exception ex)
        {
            context.Logger.LogError($"Failed to call ListBucketsAsync: {ex.Message}");
            throw;
        }
    }
}
```

```

    }
}

private async Task<ListBucketsResponse> DoListBucketsAsync()
{
    try
    {
        var putRequest = new ListBucketsRequest
        {
        };

        var response = await s3Client.ListBucketsAsync(putRequest);
        return response;
    }
    catch (AmazonS3Exception ex)
    {
        throw new Exception($"Failed to call ListBucketsAsync: {ex.Message}", ex);
    }
}
}

```

Lorsque vous invoquez cette Lambda, vous verrez la trace suivante dans la carte de suivi de la console : CloudWatch



Migrer vers OpenTelemetry Python

Ce guide vous aide à migrer les applications Python du SDK X-Ray vers l' OpenTelemetry instrumentation. Il couvre les approches d'instrumentation automatique et manuelle, avec des exemples de code pour des scénarios courants.

Sections

- [Solutions d'instrumentation automatique à code zéro](#)

- [Instrumentez manuellement vos applications](#)
- [Initialisation de la configuration de suivi](#)
- [Suivi des demandes entrantes](#)
- [AWS Instrumentation du SDK](#)
- [Instrumentation des appels HTTP sortants par le biais de requêtes](#)
- [Support d'instrumentation pour d'autres bibliothèques](#)
- [Création manuelle de données de trace](#)
- [Instrumentation Lambda](#)

Solutions d'instrumentation automatique à code zéro

Avec le SDK X-Ray, vous deviez modifier le code de votre application pour suivre les demandes. OpenTelemetry propose des solutions d'instrumentation automatique sans code pour suivre les demandes. Avec OpenTelemetry, vous avez la possibilité d'utiliser des solutions d'auto-instrumentation à code zéro pour suivre les demandes.

Code zéro avec instrumentations automatiques OpenTelemetry basées sur la base

1. Utilisation de la AWS distribution pour l'instrumentation automatique OpenTelemetry (ADOT) pour Python — Pour l'instrumentation automatique pour les applications Python, voir [Traçage et métriques avec la AWS distribution pour OpenTelemetry l'instrumentation](#) automatique de Python.

(Facultatif) Vous pouvez également activer les signaux CloudWatch d'application lorsque vous instrumentez automatiquement vos applications à l' AWS aide de l'instrumentation automatique ADOT Python pour surveiller l'état actuel des applications et suivre les performances des applications à long terme par rapport à vos objectifs commerciaux. Application Signals vous offre une vue unifiée et orientée application de vos services, dépendances et applications, et vous aide à surveiller et à trier l'état des applications.

2. Utilisation de l'instrumentation automatique à code zéro en OpenTelemetry Python — Pour l'instrumentation automatique avec OpenTelemetry Python, voir Instrumentation en [code zéro en Python](#).

Instrumentez manuellement vos applications

Vous pouvez instrumenter manuellement vos applications à l'aide de la `pip` commande.

With X-Ray SDK

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

With OpenTelemetry SDK

```
pip install opentelemetry-api opentelemetry-sdk opentelemetry-exporter-otlp  
opentelemetry-propagator-aws-xray
```

Initialisation de la configuration de suivi

With X-Ray SDK

Dans X-Ray, le global `xray_recorder` est initialisé et utilisé pour générer des segments et des sous-segments.

With OpenTelemetry SDK

Note

X-Ray Remote Sampling n'est actuellement pas disponible pour être configuré pour OpenTelemetry Python. Cependant, la prise en charge de l'échantillonnage à distance X-Ray est actuellement disponible via l'ADOT Auto-Instrumentation pour Python.

Dans OpenTelemetry, vous devez initialiser un `globalTracerProvider`. Grâce à `celatracerProvider`, vous pouvez acquérir un [traceur](#) que vous pouvez utiliser pour générer des spans n'importe où dans votre application. Il est recommandé de configurer les composants suivants :

- `OTLPSpanExporter`— Nécessaire pour exporter les traces vers l' CloudWatch OpenTelemetry agent/le collecteur
- Un propagateur de AWS rayons X : nécessaire pour propager le contexte de trace aux AWS services intégrés à X-Ray

```
from opentelemetry import (
    trace,
    propagate
)
from opentelemetry.sdk.trace import TracerProvider

from opentelemetry import trace
from opentelemetry.sdk.trace import TracerProvider
from opentelemetry.exporter.otlp.proto.http.trace_exporter import OTLPSpanExporter
from opentelemetry.sdk.trace.export import BatchSpanProcessor
from opentelemetry.propagators.aws import AwsXRayPropagator

# Sends generated traces in the OTLP format to an OTel Collector running on port
# 4318
otlp_exporter = OTLPSpanExporter(endpoint="http://localhost:4318/v1/traces")
# Processes traces in batches as opposed to immediately one after the other
span_processor = BatchSpanProcessor(otlp_exporter)
# More configurations can be done here. We will visit them later.

# Sets the global default tracer provider
provider = TracerProvider(active_span_processor=span_processor)
trace.set_tracer_provider(provider)

# Configures the global propagator to use the X-Ray Propagator
propagate.set_global_textmap(AwsXRayPropagator())

# Creates a tracer from the global tracer provider
tracer = trace.get_tracer("my.tracer.name")
# Use this tracer to create Spans
```

Avec l'auto-instrumentation ADOT pour Python

Vous pouvez utiliser l'instrumentation automatique ADOT pour Python afin de configurer automatiquement vos OpenTelemetry applications Python. En utilisant l'instrumentation automatique ADOT, vous n'avez pas besoin de modifier manuellement le code pour suivre les demandes entrantes ou pour suivre les bibliothèques telles que le AWS SDK ou les clients HTTP. Pour plus d'informations, consultez [Tracing and Metrics with the AWS Distro for OpenTelemetry Python Auto-Instrumentation](#).

L'auto-instrumentation ADOT pour Python prend en charge :

- Échantillonnage à distance X-Ray via la variable d'environnement `export OTEL_TRACES_SAMPLER=xray`
- Propagation du contexte X-Ray Trace (activée par défaut)
- Détection des ressources (la détection des ressources pour les environnements Amazon EC2, Amazon ECS et Amazon EKS est activée par défaut)
- Les instrumentations de bibliothèque automatiques pour toutes les OpenTelemetry instrumentations prises en charge sont activées par défaut. Vous pouvez le désactiver de manière sélective via la variable d'environnement `OTEL_PYTHON_DISABLED_INSTRUMENTATIONS`. (tous sont activés par défaut)
- Création manuelle de Spans

Des plug-ins du service X-Ray aux fournisseurs de OpenTelemetry AWS ressources

Le SDK X-Ray fournit des plug-ins que vous pouvez ajouter pour capturer les informations spécifiques `xray_recorder` à la plate-forme à partir du service hébergé tel qu'Amazon EC2, Amazon ECS et Elastic Beanstalk. Il est similaire aux fournisseurs de ressources en OpenTelemetry ce sens qu'il capture les informations sous forme d'attributs de ressources. Plusieurs fournisseurs de ressources sont disponibles pour différentes AWS plateformes.

- Commencez par installer le package d'AWS extension, `pip install opentelemetry-sdk-extension-aws`
- Configurez le détecteur de ressources souhaité. L'exemple suivant montre comment configurer le fournisseur de EC2 ressources Amazon dans le OpenTelemetry SDK

```
from opentelemetry import trace
from opentelemetry.sdk.trace import TracerProvider
from opentelemetry.sdk.extension.aws.resource.ec2 import (
    AwsEc2ResourceDetector,
)
from opentelemetry.sdk.resources import get_aggregated_resources

provider = TracerProvider(
    active_span_processor=span_processor,
    resource=get_aggregated_resources([
        AwsEc2ResourceDetector(),
```

```
]))
```

```
trace.set_tracer_provider(provider)
```

Suivi des demandes entrantes

With X-Ray SDK

Le SDK Python X-Ray prend en charge les frameworks d'applications tels que Django, Flask et Bottle pour suivre les demandes entrantes pour les applications Python exécutées sur ces derniers. Cela se fait en ajoutant `XRayMiddleware` à l'application pour chaque framework.

With OpenTelemetry SDK

OpenTelemetry fournit des instrumentations pour [Django](#) et [Flask](#) via les bibliothèques d'instrumentation spécifiques. Aucune instrumentation pour Bottle n'est disponible dans OpenTelemetry, les applications peuvent toujours être tracées à l'aide de l'[instrumentation OpenTelemetry WSGI](#).

Pour l'exemple de code suivant, vous avez besoin de la dépendance suivante :

```
pip install opentelemetry-instrumentation-flask
```

Vous devez initialiser le OpenTelemetry SDK et enregistrer le global `TracerProvider` avant d'ajouter des instrumentations pour votre infrastructure d'application. Sans cela, les opérations de traçage le seront `no-ops`. Une fois que vous avez configuré le `globalTracerProvider`, vous pouvez utiliser l'instrument pour votre infrastructure d'application. L'exemple suivant illustre une application Flask.

```
from flask import Flask
from opentelemetry import trace
from opentelemetry.instrumentation.flask import FlaskInstrumentor
from opentelemetry.sdk.extension.aws.resource import AwsEc2ResourceDetector
from opentelemetry.sdk.resources import get_aggregated_resources
from opentelemetry.sdk.trace import TracerProvider
from opentelemetry.sdk.trace.export import BatchSpanProcessor, ConsoleSpanExporter
```



```
provider = TracerProvider(resource=get_aggregated_resources(
    [
        AwsEc2ResourceDetector(),
    ]))

processor = BatchSpanProcessor(ConsoleSpanExporter())
provider.add_span_processor(processor)
trace.set_tracer_provider(provider)

# Creates a tracer from the global tracer provider
tracer = trace.get_tracer("my.tracer.name")

app = Flask(__name__)

# Instrument the Flask app
FlaskInstrumentor().instrument_app(app)

@app.route('/')
def hello_world():
    return 'Hello World!'

if __name__ == '__main__':
    app.run()
```

AWS Instrumentation du SDK

With X-Ray SDK

Le SDK Python de X-Ray trace la demande du client du AWS SDK en appliquant des correctifs à la bibliothèque. `botocore` Pour plus d'informations, consultez la section [Tracing AWS SDK calls with the X-Ray SDK for Python](#). Dans votre application, la `patch_all()` méthode est utilisée pour instrumenter toutes les bibliothèques ou appliquer des correctifs de manière sélective à l'aide de la `botocore` ou `boto3` des bibliothèques utilisant `patch((['botocore']))`. L'une des méthodes choisies instrumente tous les clients Boto3 de votre application et génère un sous-segment pour tout appel effectué à l'aide de ces clients.

With OpenTelemetry SDK

Pour l'exemple de code suivant, vous aurez besoin de la dépendance suivante :

```
pip install opentelemetry-instrumentation-botocore
```

Utilisez l'[instrumentation OpenTelemetry Botocore](#) de manière programmatique pour instrumenter tous les clients Boto3 de votre application. L'exemple suivant illustre l'`botocoreinstrumentation`.

```
import boto3
import opentelemetry.trace as trace
from botocore.exceptions import ClientError
from opentelemetry.sdk.trace import TracerProvider
from opentelemetry.sdk.resources import get_aggregated_resources
from opentelemetry.sdk.trace.export import (
    BatchSpanProcessor,
    ConsoleSpanExporter,
)
from opentelemetry.instrumentation.botocore import BotocoreInstrumentor

provider = TracerProvider()
processor = BatchSpanProcessor(ConsoleSpanExporter())
provider.add_span_processor(processor)
trace.set_tracer_provider(provider)

# Creates a tracer from the global tracer provider
tracer = trace.get_tracer("my.tracer.name")

# Instrument BotoCore
BotocoreInstrumentor().instrument()

# Initialize S3 client
s3 = boto3.client("s3", region_name="us-east-1")

# Your bucket name
bucket_name = "my-example-bucket"

# Get bucket location (as an example of describing it)
try:
    response = s3.get_bucket_location(Bucket=bucket_name)
    region = response.get("LocationConstraint") or "us-east-1"
    print(f"Bucket '{bucket_name}' is in region: {region}")

    # Optionally, get bucket's creation date via list_buckets
```

```
buckets = s3.list_buckets()
for bucket in buckets["Buckets"]:
    if bucket["Name"] == bucket_name:
        print(f"Bucket created on: {bucket['CreationDate']}")
        break
except ClientError as e:
    print(f"Failed to describe bucket: {e}")
```

Instrumentation des appels HTTP sortants par le biais de requêtes

With X-Ray SDK

Le SDK Python de X-Ray trace les appels HTTP sortants via les requêtes en appliquant des correctifs à la bibliothèque de requêtes. Pour plus d'informations, voir [Suivi des appels aux services Web HTTP en aval à l'aide du SDK X-Ray pour Python](#). Dans votre application, vous pouvez utiliser la `patch_all()` méthode pour instrumenter toutes les bibliothèques ou en appliquant des correctifs sélectifs aux bibliothèques de requêtes en utilisant `patch(['requests'])`. N'importe laquelle des options instrumente la `requests` bibliothèque, générant un sous-segment pour chaque appel effectué par le biais `requests` de cette option.

With OpenTelemetry SDK

Pour l'exemple de code suivant, vous aurez besoin de la dépendance suivante :

```
pip install opentelemetry-instrumentation-requests
```

Utilisez l'instrumentation OpenTelemetry des requêtes par programmation pour instrumenter la bibliothèque de requêtes afin de générer des traces pour les requêtes HTTP qu'elle effectue dans votre application. Pour plus d'informations, consultez la section [Instrumentation OpenTelemetry des requêtes](#). L'exemple suivant illustre l'instrumentation de la `requests` bibliothèque.

```
from opentelemetry.instrumentation.requests import RequestsInstrumentor

# Instrument Requests
RequestsInstrumentor().instrument()
```

```
...
```

```
example_session = requests.Session()  
example_session.get(url="https://example.com")
```

Vous pouvez également instrumenter la `urllib3` bibliothèque sous-jacente pour suivre les requêtes HTTP :

```
# pip install opentelemetry-instrumentation-urllib3  
from opentelemetry.instrumentation.urllib3 import URLLib3Instrumentor  
  
# Instrument urllib3  
URLLib3Instrumentor().instrument()  
  
...  
  
example_session = requests.Session()  
example_session.get(url="https://example.com")
```

Support d'instrumentation pour d'autres bibliothèques

Vous trouverez la liste complète des instrumentations de bibliothèque prises en charge pour OpenTelemetry Python sous [Bibliothèques, frameworks, serveurs d'applications pris en charge et JVMs](#).

Vous pouvez également effectuer une recherche OpenTelemetry dans le registre pour savoir si l'instrumentation est OpenTelemetry compatible. Consultez le [registre](#) pour commencer la recherche.

Création manuelle de données de trace

Vous pouvez créer des segments et des sous-segments en utilisant le `xray_recorder` dans votre application Python. Pour plus d'informations, consultez [Instrumentation manuelle du code Python](#). Vous pouvez également ajouter manuellement des annotations et des métadonnées aux données de trace.

Création de spans avec le SDK OpenTelemetry

Utilisez l'`start_as_current_span` API pour démarrer un intervalle et configurez-le pour créer des intervalles. Pour des exemples de création de travées, voir [Création de travées](#). Une fois qu'une période est démarrée et qu'elle est dans son étendue actuelle, vous pouvez y ajouter des informations supplémentaires en ajoutant des attributs, des événements, des exceptions, des liens, etc. Tout comme nous avons des segments et des sous-segments dans X-Ray, il existe différents types de travées. OpenTelemetry Seules les plages de SERVER type sont converties en segments X-Ray, tandis que les autres sont converties en sous-segments de X-Ray.

```
from opentelemetry import trace
from opentelemetry.trace import SpanKind

import time

tracer = trace.get_tracer("my.tracer.name")

# Create a new span to track some work
with tracer.start_as_current_span("parent", kind=SpanKind.SERVER) as parent_span:
    time.sleep(1)

    # Create a nested span to track nested work
    with tracer.start_as_current_span("child", kind=SpanKind.CLIENT) as child_span:
        time.sleep(2)
        # the nested span is closed when it's out of scope

    # Now the parent span is the current span again
    time.sleep(1)

# This span is also closed when it goes out of scope
```

Ajouter des annotations et des métadonnées aux traces avec OpenTelemetry le SDK

Le SDK X-Ray Python fournit une solution séparée APIs permettant d'ajouter `put_metadata` des annotations et des métadonnées à une trace. `put_annotation` Dans le OpenTelemetry SDK, les annotations et les métadonnées sont simplement des attributs sur une plage, ajoutés via l'`set_attribute` API.

Les attributs Span que vous souhaitez utiliser comme annotations sur une trace sont ajoutés sous la clé réservée `aws.xray.annotations` dont la valeur est une liste de paires clé-valeur

d'annotations. Tous les autres attributs span deviennent des métadonnées sur le segment ou le sous-segment converti.

De plus, si vous utilisez le collecteur ADOT, vous pouvez configurer les attributs de span à convertir en annotations X-Ray en les spécifiant `indexed_attributes` dans la configuration du collecteur.

L'exemple ci-dessous montre comment ajouter des annotations et des métadonnées à une trace à l'aide du OpenTelemetry SDK.

```
with tracer.start_as_current_span("parent", kind=SpanKind.SERVER) as parent_span:
    parent_span.set_attribute("TransactionId", "qwerty12345")
    parent_span.set_attribute("AccountId", "1234567890")

    # This will convert the TransactionId and AccountId to be searchable X-Ray
    annotations
    parent_span.set_attribute("aws.xray.annotations", ["TransactionId", "AccountId"])

    with tracer.start_as_current_span("child", kind=SpanKind.CLIENT) as child_span:

        # The MicroTransactionId will be converted to X-Ray metadata for the child
        subsegment
        child_span.set_attribute("MicroTransactionId", "micro12345")
```

Instrumentation Lambda

Pour surveiller vos fonctions Lambda sur X-Ray, vous avez activé X-Ray et ajouté les autorisations appropriées au rôle d'invocation des fonctions. De plus, si vous suivez les requêtes en aval de votre fonction, vous instrumenterez le code avec le SDK X-Ray Python.

OpenTelemetry Pour X-Ray, il est recommandé d'utiliser la couche lambda des signaux CloudWatch d'application lorsque les [signaux d'application](#) sont désactivés. Cela instrumentera automatiquement votre fonction et générera des intervalles pour l'invocation de la fonction et pour toute demande en aval émanant de votre fonction. Outre le suivi, si vous souhaitez utiliser les signaux d'application pour surveiller l'état de votre fonction, consultez [Activer vos applications sur Lambda](#).

- Trouvez l'ARN de couche Lambda requis pour votre fonction dans [AWS Lambda Layer for OpenTelemetry ARNs](#) et ajoutez-le.
- Définissez les variables d'environnement suivantes pour votre fonction.

- `AWS_LAMBDA_EXEC_WRAPPER=/opt/otel-instrument`— Cela charge l'auto-instrumentation pour la fonction
- `OTEL_AWS_APPLICATION_SIGNALS_ENABLED=false`— Cela désactivera la surveillance des signaux d'application

Création manuelle de travées avec l'instrumentation Lambda

En outre, vous pouvez générer des intervalles personnalisés au sein de votre fonction pour suivre le travail. Vous pouvez le faire en utilisant uniquement le `opentelemetry-api` package conjointement avec l'instrumentation automatique de la couche Lambda d'Application Signals.

1. Incluez le `opentelemetry-api` en tant que dépendance dans votre fonction
2. L'extrait de code suivant est un exemple permettant de générer des étendues personnalisées

```
from opentelemetry import trace

# Get the tracer (auto-configured by the Application Signals layer)
tracer = trace.get_tracer(__name__)

def handler(event, context):
    # This span is a child of the layer's root span
    with tracer.start_as_current_span("my-custom-span") as span:
        span.set_attribute("key1", "value1")
        span.add_event("custom-event", {"detail": "something happened"})

        # Any logic you want to trace
        result = some_internal_logic()

    return {
        "statusCode": 200,
        "body": result
    }
```

Migrer vers OpenTelemetry Ruby

Pour migrer vos applications Ruby du SDK X-Ray vers l' OpenTelemetry instrumentation, utilisez les exemples de code et les instructions suivants pour l'instrumentation manuelle.

Sections

- [Instrumentez manuellement vos solutions avec le SDK](#)
- [Suivi des demandes entrantes \(instrumentation Rails\)](#)
- [AWS Instrumentation du SDK](#)
- [Instrumentation des appels HTTP sortants](#)
- [Support d'instrumentation pour d'autres bibliothèques](#)
- [Création manuelle de données de trace](#)
- [Instrumentation manuelle Lambda](#)

Instrumentez manuellement vos solutions avec le SDK

Tracing setup with X-Ray SDK

Le SDK X-Ray pour Ruby nécessitait que vous configuriez votre code à l'aide de plug-ins de service.

```
require 'aws-xray-sdk'

XRay.recorder.configure(plugins: [:ec2, :elastic_beanstalk])
```

Tracing setup with OpenTelemetry SDK

Note

L'échantillonnage à distance X-Ray n'est actuellement pas disponible pour être configuré pour OpenTelemetry Ruby.

Pour une application Ruby on Rails, placez votre code de configuration dans un initialiseur Rails. Pour plus d'informations, consultez [Mise en route avec](#) . Pour tous les programmes Ruby instrumentés manuellement, vous devez utiliser la `OpenTelemetry::SDK.configure` méthode pour configurer le SDK OpenTelemetry Ruby.

Installez d'abord les packages suivants :

```
bundle add opentelemetry-sdk opentelemetry-exporter-otlp opentelemetry-propagator-xray
```


Configurez ensuite le OpenTelemetry SDK via le code de configuration qui s'exécute lors de l'initialisation de votre programme. Il est recommandé de configurer les composants suivants :

- **OTLP Exporter**— Nécessaire pour exporter les traces vers l' CloudWatch agent et le OpenTelemetry collecteur
- **An AWS X-Ray Propagator**— Nécessaire pour propager le contexte de trace aux AWS services intégrés à X-Ray

```
require 'opentelemetry-sdk'
require 'opentelemetry-exporter-otlp'

# Import the gem containing the AWS X-Ray for OTel Ruby ID Generator and propagator
require 'opentelemetry-propagator-xray'

OpenTelemetry::SDK.configure do |c|
  c.service_name = 'my-service-name'

  c.add_span_processor(
    # Use the BatchSpanProcessor to send traces in groups instead of one at a time
    OpenTelemetry::SDK::Trace::Export::BatchSpanProcessor.new(
      # Use the default OTLP Exporter to send traces to the ADOT Collector
      OpenTelemetry::Exporter::OTLP::Exporter.new(
        # The OpenTelemetry Collector is running as a sidecar and listening on port
4318
        endpoint:"http://127.0.0.1:4318/v1/traces"
      )
    )
  )

  # The X-Ray Propagator injects the X-Ray Tracing Header into downstream calls
  c.propagators = [OpenTelemetry::Propagator::XRay::TextMapPropagator.new]
end
```

OpenTelemetry SDKs ont également le concept d'instrumentations de bibliothèque. Leur activation créera automatiquement des étendues pour les bibliothèques telles que le AWS SDK. OpenTelemetry offre la possibilité d'activer toutes les instrumentations de bibliothèque ou de spécifier les instrumentations de bibliothèque à activer.

Pour activer toutes les instrumentations, installez d'abord le `opentelemetry-instrumentation-all` package :

```
bundle add opentelemetry-instrumentation-all
```

Ensuite, mettez à jour la configuration pour activer toutes les instrumentations de bibliothèque, comme indiqué ci-dessous :

```
require 'opentelemetry/instrumentation/all'
...

OpenTelemetry::SDK.configure do |c|
  ...

  c.use_all() # Enable all instrumentations
end
```

OpenTelemetry SDKs ont également le concept d'instrumentations de bibliothèque. Leur activation créera automatiquement des étendues pour les bibliothèques telles que le AWS SDK. OpenTelemetry offre la possibilité d'activer toutes les instrumentations de bibliothèque ou de spécifier les instrumentations de bibliothèque à activer.

Pour activer toutes les instrumentations, installez d'abord le `opentelemetry-instrumentation-all` package :

```
bundle add opentelemetry-instrumentation-all
```

Ensuite, mettez à jour la configuration pour activer toutes les instrumentations de bibliothèque, comme indiqué ci-dessous :

```
require 'opentelemetry/instrumentation/all'
...

OpenTelemetry::SDK.configure do |c|
  ...

  c.use_all() # Enable all instrumentations
```

```
end
```

Suivi des demandes entrantes (instrumentation Rails)

With X-Ray SDK

Avec le SDK X-Ray, le traçage X-Ray est configuré pour le framework Rails lors de l'initialisation.

Exemple — config/initializers/aws_xray.rb

```
Rails.application.config.xray = {  
  name: 'my app',  
  patch: %I[net_http aws_sdk],  
  active_record: true  
}
```

With OpenTelemetry SDK

Installez d'abord les packages suivants :

```
bundle add opentelemetry-instrumentation-rack opentelemetry-instrumentation-rails opentelemetry-instrumentation-action_pack opentelemetry-instrumentation-active_record opentelemetry-instrumentation-action_view
```

Ensuite, mettez à jour la configuration pour activer l'instrumentation pour votre application Rails, comme indiqué ci-dessous :

```
# During SDK configuration  
OpenTelemetry::SDK.configure do |c|  
  
  ...  
  
  c.use 'OpenTelemetry::Instrumentation::Rails'  
  c.use 'OpenTelemetry::Instrumentation::Rack'  
  c.use 'OpenTelemetry::Instrumentation::ActionPack'  
  c.use 'OpenTelemetry::Instrumentation::ActiveSupport'  
  c.use 'OpenTelemetry::Instrumentation::ActionView'
```

```
...  
end
```

AWS Instrumentation du SDK

With X-Ray SDK

Pour traiter les AWS demandes sortantes du AWS SDK, les clients du AWS SDK sont corrigés avec X-Ray, comme dans l'exemple suivant :

```
require 'aws-xray-sdk'  
require 'aws-sdk-s3'  
  
# Patch AWS SDK clients  
XRay.recorder.configure(plugins: [:aws_sdk])  
  
# Use the instrumented client  
s3 = Aws::S3::Client.new  
s3.list_buckets
```

With OpenTelemetry SDK

AWS Le SDK pour Ruby V3 prend en charge l'enregistrement et OpenTelemetry l'émission de traces. Pour plus d'informations sur la configuration d'un client de service, consultez la [section Configuration des fonctionnalités d'observabilité dans le AWS SDK OpenTelemetry](#) pour Ruby.

Instrumentation des appels HTTP sortants

Lorsque vous effectuez des appels HTTP vers des services externes, vous devrez peut-être instrumenter les appels manuellement si l'instrumentation automatique n'est pas disponible ou ne fournit pas suffisamment de détails.

With X-Ray SDK

Pour instrumenter les appels en aval, le SDK X-Ray pour Ruby a été utilisé pour patcher `net/http` la bibliothèque utilisée par votre application :

```
require 'aws-xray-sdk'

config = {
  name: 'my app',
  patch: %I[net_http]
}

XRay.recorder.configure(config)
```

With OpenTelemetry SDK

Pour activer l'`net/http` instrumentation à l'aide de OpenTelemetry, installez d'abord le `opentelemetry-instrumentation-net_http` package :

```
bundle add opentelemetry-instrumentation-net_http
```

Ensuite, mettez à jour la configuration pour activer l'`net/http` instrumentation comme indiqué ci-dessous :

```
OpenTelemetry::SDK.configure do |c|
  ...

  c.use 'OpenTelemetry::Instrumentation::Net::HTTP'
  ...
end
```

Support d'instrumentation pour d'autres bibliothèques

Vous trouverez la liste complète des instrumentations de bibliothèque prises en charge pour OpenTelemetry Ruby sous [opentelemetry-ruby-contrib](#).

Vous pouvez également effectuer une recherche OpenTelemetry dans le registre pour savoir si l'instrumentation est OpenTelemetry compatible. Pour plus d'informations, consultez la section [Registre](#).

Création manuelle de données de trace

With X-Ray SDK

À l'aide de X-Ray, le `aws-xray-sdk` package vous a demandé de créer manuellement des segments et leurs sous-segments secondaires pour suivre votre application. Vous avez peut-être également ajouté des annotations et des métadonnées X-Ray à vos segments ou sous-segments :

```
require 'aws-xray-sdk'
...

# Start a segment
segment = XRay.recorder.begin_segment('my-service')

# Add annotations (indexed key-value pairs)
segment.annotations[:user_id] = 'user-123'
segment.annotations[:payment_status] = 'completed'

# Add metadata (non-indexed data)
segment.metadata[:order] = {
  id: 'order-456',
  items: [
    { product_id: 'prod-1', quantity: 2 },
    { product_id: 'prod-2', quantity: 1 }
  ],
  total: 67.99
}

# Add metadata to a specific namespace
segment.metadata(namespace: 'payment') do |metadata|
  metadata[:transaction_id] = 'tx-789'
  metadata[:payment_method] = 'credit_card'
end

# Create a subsegment with annotations and metadata
segment.subsegment('payment-processing') do |subsegment1|
  subsegment1.annotations[:payment_id] = 'pay-123'
  subsegment1.metadata[:details] = { amount: 67.99, currency: 'USD' }

  # Create a nested subsegment
  subsegment1.subsegment('operation-2') do |subsegment2|
    # Do more work...
  end
end
```

```
    end
  end

  # Close the segment
  segment.close
```

With OpenTelemetry SDK

Vous pouvez utiliser des intervalles personnalisés pour surveiller les performances des activités internes qui ne sont pas capturées par les bibliothèques d'instrumentation. Notez que seules les plages du type `server` sont converties en segments X-Ray, toutes les autres plages sont converties en sous-segments X-Ray. Par défaut, les intervalles sont `INTERNAL`.

Tout d'abord, créez un traceur afin de générer des spans, que vous pouvez obtenir grâce à la `OpenTelemetry.tracer_provider.tracer('<YOUR_TRACER_NAME>')` méthode. Cela fournira une instance `Tracer` enregistrée globalement dans la `OpenTelemetry` configuration de votre application. Il est courant d'avoir un seul traceur pour l'ensemble d'une application. Créez un `OpenTelemetry` traceur et utilisez-le pour créer des travées :

```
require 'opentelemetry-sdk'

...

# Get a tracer
tracer = OpenTelemetry.tracer_provider.tracer('my-application')

# Create a server span (equivalent to X-Ray segment)
tracer.in_span('my-application', kind: OpenTelemetry::Trace::SpanKind::SERVER) do |span|
  # Do work...

  # Create nested spans of default kind INTERNAL will become an X-Ray subsegment
  tracer.in_span('operation-1') do |child_span1|
    # Set attributes (equivalent to X-Ray annotations and metadata)
    child_span1.set_attribute('key', 'value')

    # Do more work...
    tracer.in_span('operation-2') do |child_span2|
      # Do more work...
    end
  end
end
```

```
end
```

Ajouter des annotations et des métadonnées aux traces avec OpenTelemetry le SDK

Utilisez `set_attribute` cette méthode pour ajouter des attributs à chaque intervalle. Notez que par défaut, tous ces attributs d'étendue seront convertis en métadonnées dans les données brutes de X-Ray. Pour garantir qu'un attribut est converti en annotation et non en métadonnées, vous pouvez ajouter cette clé d'`aws.xray.annotationsattribut` à la liste des attributs. Pour plus d'informations, consultez [Enable The Customized X-Ray Annotations](#).

```
# SERVER span will become an X-Ray segment
tracer.in_span('my-server-operation', kind: OpenTelemetry::Trace::SpanKind::SERVER)
do |span|
  # Your server logic here
  span.set_attribute('attribute.key', 'attribute.value')
  span.set_attribute("metadataKey", "metadataValue")
  span.set_attribute("annotationKey1", "annotationValue")

  # Create X-Ray annotations
  span.set_attribute("aws.xray.annotations", ["annotationKey1"])
end
```

Instrumentation manuelle Lambda

With X-Ray SDK

Une fois Active Tracing activé sur Lambda, aucune configuration supplémentaire n'est requise pour utiliser le SDK X-Ray. Lambda créera un segment représentant l'invocation du gestionnaire Lambda, et vous pouvez créer des sous-segments ou des bibliothèques d'instruments à l'aide du SDK X-Ray sans aucune configuration supplémentaire.

With OpenTelemetry SDK

Examinez l'exemple de code de fonction Lambda suivant (sans instrumentation) :

```
require 'json'
def lambda_handler(event:, context:)
  # TODO implement
  { statusCode: 200, body: JSON.generate('Hello from Lambda!') }
```



```
end
```

Pour instrumenter manuellement votre Lambda, vous devez :

1. Ajoutez les gemmes suivantes pour votre Lambda

```
gem 'opentelemetry-sdk'  
gem 'opentelemetry-exporter-otlp'  
gem 'opentelemetry-propagator-xray'  
gem 'aws-distro-opentelemetry-exporter-xray-udp'  
gem 'opentelemetry-instrumentation-aws_lambda'  
gem 'opentelemetry-propagator-xray', '~> 0.24.0' # Requires version v0.24.0 or  
higher
```

2. Initialisez le OpenTelemetry SDK en dehors de votre gestionnaire Lambda. Il est recommandé de configurer le OpenTelemetry SDK avec :

1. Un processeur span simple doté d'un exportateur de span UDP X-Ray pour envoyer des traces au point de terminaison UDP X-Ray de Lambda
2. Un propagateur X-Ray Lambda
3. `service_name` configuration à définir sur le nom de la fonction Lambda

3. Dans votre classe de gestionnaire Lambda, ajoutez les lignes suivantes pour instrumenter votre gestionnaire Lambda :

```
class Handler  
  extend OpenTelemetry::Instrumentation::AwsLambda::Wrap  
  ...  
  
  instrument_handler :process  
end
```

Le code suivant illustre la fonction Lambda après les modifications requises. Vous pouvez créer des travées personnalisées supplémentaires pour compléter les travées fournies automatiquement.

```
require 'json'  
require 'opentelemetry-sdk'
```

```

require 'aws/distro/opentelemetry/exporter/xray/udp'
require 'opentelemetry/propagator/xray'
require 'opentelemetry/instrumentation/aws_lambda'

# Initialize OpenTelemetry SDK outside handler
OpenTelemetry::SDK.configure do |c|
  # Configure the AWS Distro for OpenTelemetry X-Ray Lambda exporter
  c.add_span_processor(
    OpenTelemetry::SDK::Trace::Export::SimpleSpanProcessor.new(
      AWS::Distro::OpenTelemetry::Exporter::XRay::UDP::AWSXRayUDPSpanExporter.new
    )
  )

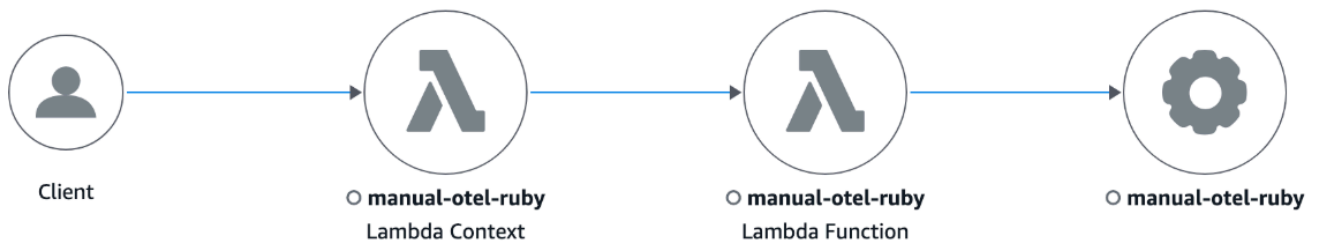
  # Configure X-Ray Lambda propagator
  c.propagators = [OpenTelemetry::Propagator::XRay.lambda_text_map_propagator]

  # Set minimal resource information
  c.resource = OpenTelemetry::SDK::Resources::Resource.create({
    OpenTelemetry::SemanticConventions::Resource::SERVICE_NAME =>
ENV['AWS_LAMBDA_FUNCTION_NAME']
  })
  c.use 'OpenTelemetry::Instrumentation::AwsLambda'
end

module LambdaFunctions
  class Handler
    extend OpenTelemetry::Instrumentation::AwsLambda::Wrap
    def self.process(event:, context:)
      "Hello!"
    end
    instrument_handler :process
  end
end

```

Voici un exemple de carte de trace d'une fonction Lambda instrumentée écrite en Ruby.



Vous pouvez également utiliser des couches Lambda OpenTelemetry pour configurer votre Lambda. Pour plus d'informations, consultez [OpenTelemetry AWS-Lambda Instrumentation](#).

Création de ressources X-Ray avec AWS CloudFormation

AWS X-Ray est intégré à AWS CloudFormation un service qui vous aide à modéliser et à configurer vos AWS ressources afin que vous puissiez passer moins de temps à créer et à gérer vos ressources et votre infrastructure. Vous créez un modèle qui décrit toutes les AWS ressources que vous souhaitez, et qui AWS CloudFormation fournit et configure ces ressources pour vous.

Lorsque vous l'utilisez AWS CloudFormation, vous pouvez réutiliser votre modèle pour configurer vos ressources X-Ray de manière cohérente et répétée. Décrivez vos ressources une seule fois, puis fournissez les mêmes ressources encore et encore dans plusieurs Comptes AWS régions.

X-Ray et AWS CloudFormation modèles

Pour fournir et configurer des ressources pour X-Ray et les services associés, vous devez comprendre les [AWS CloudFormation modèles](#). Les modèles sont des fichiers texte formatés en JSON ou YAML. Ces modèles décrivent les ressources que vous souhaitez mettre à disposition dans vos AWS CloudFormation piles. Si vous n'êtes pas familiarisé avec JSON ou YAML, vous pouvez utiliser AWS CloudFormation Designer pour vous aider à démarrer avec les AWS CloudFormation modèles. Pour plus d'informations, consultez [Qu'est-ce que AWS CloudFormation Designer ?](#) dans le AWS CloudFormation Guide de l'utilisateur.

X-Ray soutient [AWS::XRay::Group](#) la [AWS::XRay::SamplingRule](#) création et les [AWS::XRay::ResourcePolicy](#) ressources dans AWS CloudFormation. Pour plus d'informations, notamment des exemples de modèles JSON et YAML, consultez la [référence au type de ressource X-Ray](#) dans le guide de l'AWS CloudFormation utilisateur.

En savoir plus sur AWS CloudFormation

Pour en savoir plus AWS CloudFormation, consultez les ressources suivantes :

- [AWS CloudFormation](#)
- [AWS CloudFormation Guide de l'utilisateur](#)
- [AWS CloudFormation API Reference](#)
- [AWS CloudFormation Guide de l'utilisateur de l'interface de ligne de commande](#)

Étiquetage des règles et des groupes d'échantillonnage aux rayons X

Les balises sont des mots ou des phrases que vous pouvez utiliser pour identifier et organiser vos AWS ressources. Vous pouvez ajouter plusieurs balises à chaque ressource. Chaque balise inclut une clé et une valeur facultative que vous définissez. Par exemple, une clé de balise peut être **domain**, et la valeur de la balise peut être **example.com**. Vous pouvez rechercher et filtrer vos ressources en fonction des balises que vous ajoutez. Pour plus d'informations sur les méthodes d'utilisation des balises, consultez la section [AWS Ressources de balisage](#) dans le manuel de référence AWS général.

Vous pouvez utiliser des balises pour appliquer des autorisations basées sur des balises aux CloudFront distributions. Pour plus d'informations, consultez la section [Contrôle de l'accès aux AWS ressources à l'aide de balises de ressources](#).

Note

[Tag Editor](#) et [AWS Resource Groups](#) ne prennent actuellement pas en charge les ressources X-Ray. Vous pouvez ajouter et gérer des balises à l'aide de la AWS X-Ray console ou de l'API.

Vous pouvez appliquer des balises aux ressources à l'aide de la console X-Ray, de l'API AWS CLI SDKs, et AWS Tools for Windows PowerShell. Pour plus d'informations, consultez la documentation de suivante :

- X-Ray API — Consultez les opérations suivantes dans la référence de l'AWS X-Ray API :
 - [ListTagsForResource](#)
 - [CreateSamplingRule](#)
 - [CreateGroup](#)
 - [TagResource](#)
 - [UntagResource](#)
- AWS CLI — Voir [xray dans le manuel](#) de référence des AWS CLI commandes
- SDKs — Consultez la documentation du SDK applicable sur la page de [AWS documentation](#)

 Note

Si vous ne pouvez pas ajouter ou modifier des balises sur une ressource X-Ray, ou si vous ne pouvez pas ajouter une ressource dotée de balises spécifiques, vous n'êtes peut-être pas autorisé à effectuer cette opération. Pour demander l'accès, contactez un AWS utilisateur de votre entreprise qui possède des autorisations d'administrateur dans X-Ray.

Rubriques

- [Restrictions liées aux étiquettes](#)
- [Gestion des tags dans la console](#)
- [Gestion des balises dans AWS CLI](#)
- [Contrôlez l'accès aux ressources X-Ray en fonction des balises](#)

Restrictions liées aux étiquettes

Les restrictions suivantes s'appliquent aux balises :

- Nombre maximal de balises par ressource – 50
- Longueur de clé maximale – 128 caractères Unicode
- Longueur de valeur maximale – 256 caractères Unicode
- Caractères acceptés pour les clés et valeurs – a-z, A-Z, 0-9, espace et les caractères suivants :
_ . : / = + - and @
- Les clés et valeurs de balise sont sensibles à la casse.
- Ne l'utilisez pas aws : comme préfixe pour les clés ; il est réservé à l' AWS usage.

 Note

Vous ne pouvez pas modifier ou supprimer les balises système.

Gestion des tags dans la console

Vous pouvez ajouter des balises facultatives lorsque vous créez un groupe X-Ray ou une règle d'échantillonnage. Les balises peuvent également être modifiées ou supprimées ultérieurement dans la console.

Les procédures suivantes expliquent comment ajouter, modifier et supprimer des balises pour vos groupes et des règles d'échantillonnage dans la console X-Ray.

Rubriques

- [Ajouter des tags à un nouveau groupe \(console\)](#)
- [Ajouter des balises à une nouvelle règle d'échantillonnage \(console\)](#)
- [Modifier ou supprimer les balises d'un groupe \(console\)](#)
- [Modifier ou supprimer des balises pour une règle d'échantillonnage \(console\)](#)

Ajouter des tags à un nouveau groupe (console)

Lorsque vous créez un nouveau groupe X-Ray, vous pouvez ajouter des balises facultatives sur la page Create group.

1. Connectez-vous à la console X-Ray AWS Management Console et ouvrez-la <https://console.aws.amazon.com/xray/chez> vous.
2. Dans le volet de navigation, développez Configuration, puis choisissez Groups.
3. Choisissez Créer un groupe.
4. Sur la page Créer un groupe, spécifiez un nom et une expression de filtre pour le groupe. Pour de plus amples informations sur ces propriétés, veuillez consulter [Configuration des groupes](#).
5. Dans Balises, entrez une clé de balise et, éventuellement, une valeur de balise. Par exemple, vous pouvez entrer une clé de **Stage** balise et une valeur de balise de **Production** pour indiquer que ce groupe est destiné à une utilisation en production. Lorsque vous ajoutez un tag, une nouvelle ligne apparaît pour vous permettre d'ajouter un autre tag, si nécessaire. Consultez [Restrictions liées aux étiquettes](#) cette rubrique pour connaître les restrictions relatives aux balises.
6. Lorsque vous avez terminé d'ajouter des balises, choisissez Create group.

Ajouter des balises à une nouvelle règle d'échantillonnage (console)

Lorsque vous créez une nouvelle règle d'échantillonnage X-Ray, vous pouvez ajouter des balises sur la page Créer une règle d'échantillonnage.

1. Connectez-vous à la console X-Ray AWS Management Console et ouvrez-la <https://console.aws.amazon.com/xray/chez> vous.
2. Dans le volet de navigation, développez Configuration, puis choisissez Sampling.
3. Choisissez Créer une règle d'échantillonnage.
4. Sur la page Créer une règle d'échantillonnage, spécifiez un nom, une priorité, des limites, des critères de correspondance et des attributs correspondants. Pour de plus amples informations sur ces propriétés, veuillez consulter [Configuration de règles d'échantillonnage](#).
5. Dans Balises, entrez une clé de balise et, éventuellement, une valeur de balise. Par exemple, vous pouvez entrer une clé de **Stage** balise et une valeur de balise de **Production** pour indiquer que cette règle d'échantillonnage est destinée à une utilisation en production. Lorsque vous ajoutez un tag, une nouvelle ligne apparaît pour vous permettre d'ajouter un autre tag, si nécessaire. Consultez [Restrictions liées aux étiquettes](#) cette rubrique pour connaître les restrictions relatives aux balises.
6. Lorsque vous avez terminé d'ajouter des balises, choisissez Créer une règle d'échantillonnage.

Modifier ou supprimer les balises d'un groupe (console)

Vous pouvez modifier ou supprimer les balises d'un groupe X-Ray sur la page Modifier le groupe.

1. Connectez-vous à la console X-Ray AWS Management Console et ouvrez-la <https://console.aws.amazon.com/xray/chez> vous.
2. Dans le volet de navigation, développez Configuration, puis choisissez Groups.
3. Dans le tableau Groupes, choisissez le nom d'un groupe.
4. Sur la page Modifier le groupe, dans Balises, modifiez les clés et les valeurs des balises. Vous ne pouvez pas avoir de clés de tag dupliquées. Les valeurs des balises sont facultatives ; vous pouvez supprimer des valeurs si vous le souhaitez. Pour plus d'informations sur les autres propriétés sur la page Modifier un groupe, consultez [Configuration des groupes](#). Consultez [Restrictions liées aux étiquettes](#) cette rubrique pour connaître les restrictions relatives aux balises.
5. Pour supprimer un tag, choisissez X à droite du tag.

6. Lorsque vous avez terminé de modifier ou de supprimer des balises, choisissez Mettre à jour le groupe.

Modifier ou supprimer des balises pour une règle d'échantillonnage (console)

Vous pouvez modifier ou supprimer les balises d'une règle d'échantillonnage X-Ray sur la page Modifier la règle d'échantillonnage.

1. Connectez-vous à la console X-Ray AWS Management Console et ouvrez-la <https://console.aws.amazon.com/xray/chez> vous.
2. Dans le volet de navigation, développez Configuration, puis choisissez Sampling.
3. Dans le tableau Règles d'échantillonnage, choisissez le nom d'une règle d'échantillonnage.
4. Dans Balises, modifiez les clés et les valeurs des balises. Vous ne pouvez pas avoir de clés de tag dupliquées. Les valeurs des balises sont facultatives ; vous pouvez supprimer des valeurs si vous le souhaitez. Pour plus d'informations sur les autres propriétés sur la page Modifier la règle d'échantillonnage, consultez [Configuration de règles d'échantillonnage](#) . Consultez [Restrictions liées aux étiquettes](#) cette rubrique pour connaître les restrictions relatives aux balises.
5. Pour supprimer un tag, choisissez X à droite du tag.
6. Lorsque vous avez terminé de modifier ou de supprimer des balises, choisissez Mettre à jour la règle d'échantillonnage.

Gestion des balises dans AWS CLI

Vous pouvez ajouter des balises lorsque vous créez un groupe X-Ray ou une règle d'échantillonnage. Vous pouvez également utiliser le AWS CLI pour créer et gérer des balises. Pour mettre à jour les balises d'un groupe existant ou d'une règle d'échantillonnage, utilisez la AWS X-Ray console ou le [TagResource](#) ou [UntagResource](#) APIs.

Rubriques

- [Ajouter des balises à un nouveau groupe X-Ray ou à une nouvelle règle d'échantillonnage \(CLI\)](#)
- [Ajouter des balises à une ressource existante \(CLI\)](#)
- [Répertorier les balises d'une ressource \(CLI\)](#)
- [Supprimer des balises sur une ressource \(CLI\)](#)

Ajouter des balises à un nouveau groupe X-Ray ou à une nouvelle règle d'échantillonnage (CLI)

Pour ajouter des balises facultatives lors de la création d'un nouveau groupe X-Ray ou d'une nouvelle règle d'échantillonnage, utilisez l'une des commandes suivantes.

- Pour ajouter des balises à un nouveau groupe, exécutez la commande suivante, en les *group_name* remplaçant par le nom de votre groupe, *mydomain.com* par le point de terminaison de votre service, *key_name* par une clé de balise et éventuellement *value* par une valeur de balise. Pour plus d'informations sur la création d'un groupe, consultez [Groups](#).

```
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"}]
```

Voici un exemple.

```
aws xray create-group \  
  --group-name "AdminGroup" \  
  --filter-expression "service(\"mydomain.com\") {fault OR error}" \  
  --tags [{"Key": "Stage", "Value": "Prod"}, {"Key": "Department", "Value": "QA"}]
```

- Pour ajouter des balises à une nouvelle règle d'échantillonnage, exécutez la commande suivante, en les *key_name* remplaçant par une clé de balise et éventuellement *value* par une valeur de balise. Cette commande spécifie les valeurs du `--sampling-rule` paramètre sous forme de fichier JSON. Pour plus d'informations sur la création d'une règle d'échantillonnage, consultez [Règles d'échantillonnage](#).

```
aws xray create-sampling-rule \  
  --cli-input-json file://file_name.json
```

Le contenu du fichier *file_name.json* JSON spécifié par le `--cli-input-json` paramètre est le suivant.

```
{  
  "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"
    }
  ]
}

```

Voici un exemple de commande.

```

aws xray create-sampling-rule \
  --cli-input-json file://9000-base-scorekeep.json

```

Le contenu du `9000-base-scorekeep.json` fichier d'exemple spécifié par le `--cli-input-json` paramètre est le suivant.

```

{
  "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"
    }
  ]
}
```

Ajouter des balises à une ressource existante (CLI)

Vous pouvez exécuter la `tag-resource` commande pour ajouter des balises à un groupe X-Ray existant ou à une règle d'échantillonnage. Cette méthode peut être plus simple que d'ajouter des balises en exécutant `update-group` ou `update-sampling-rule`.

Pour ajouter des balises à un groupe ou à une règle d'échantillonnage, exécutez la commande suivante en remplaçant l'ARN par l'ARN de la ressource et en spécifiant les clés et les valeurs facultatives des balises que vous souhaitez ajouter.

```
aws xray tag-resource \
  --resource-arn "ARN" \
  --tag-keys [{"Key": "key_name", "Value": "value"}, {"Key": "key_name", "Value": "value"}]
```

Voici un exemple.

```
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"}]
```

Répertorier les balises d'une ressource (CLI)

Vous pouvez exécuter la `list-tags-for-resource` commande pour répertorier les balises d'un groupe X-Ray ou d'une règle d'échantillonnage.

Pour répertorier les balises associées à un groupe ou à une règle d'échantillonnage, exécutez la commande suivante en remplaçant l'ARN par l'ARN de la ressource.

```
aws xray list-tags-for-resource \  
  --resource-arn "ARN"
```

Voici un exemple.

```
aws xray list-tags-for-resource \  
  --resource-arn "arn:aws:xray:us-east-2:01234567890:group/AdminGroup"
```

Supprimer des balises sur une ressource (CLI)

Vous pouvez exécuter la `untag-resource` commande pour supprimer des balises d'un groupe X-Ray ou d'une règle d'échantillonnage.

Pour supprimer des balises d'un groupe ou d'une règle d'échantillonnage, exécutez la commande suivante en remplaçant l'ARN par l'ARN de la ressource et en spécifiant les clés des balises que vous souhaitez supprimer.

Vous ne pouvez supprimer que des balises entières à l'aide de la `untag-resource` commande. Pour supprimer des valeurs de balises, utilisez la console X-Ray ou supprimez des balises et ajoutez de nouvelles balises avec les mêmes clés, mais des valeurs différentes ou vides.

```
aws xray untag-resource \  
  --resource-arn "ARN" \  
  --tag-keys ["key_name", "key_name"]
```

Voici un exemple.

```
aws xray untag-resource \  
  --resource-arn "arn:aws:xray:us-east-2:01234567890:group/group_name" \  
  --tag-keys ["Stage", "Department"]
```

Contrôlez l'accès aux ressources X-Ray en fonction des balises

Vous pouvez associer des tags à des groupes X-Ray ou à des règles d'échantillonnage, ou transmettre des tags dans une demande à X-Ray. Pour contrôler l'accès basé sur des étiquettes,

vous devez fournir les informations d'étiquette dans l'[élément de condition](#) d'une politique utilisant les clés de condition `xray:ResourceTag/key-name`, `aws:RequestTag/key-name` ou `aws:TagKeys`. Pour en savoir plus sur ces clés de condition, consultez la section [Contrôle de l'accès aux AWS ressources à l'aide de balises de ressources](#).

Pour visualiser un exemple de politique basée sur l'identité permettant de limiter l'accès à une ressource en fonction des balises de cette ressource, consultez [Gestion de l'accès aux groupes X-Ray et des règles d'échantillonnage en fonction des balises](#).

Résolution des problèmes AWS X-Ray

Cette rubrique répertorie les erreurs et les problèmes courants que vous pouvez rencontrer lors de l'utilisation de l'API, de la console ou de la console X-Ray SDKs. Si vous rencontrez un problème qui n'est pas répertorié ici, vous pouvez utiliser le bouton Commentaire sur cette page pour le signaler.

Sections

- [Carte de traçage X-Ray et pages de détails des traces](#)
- [Kit de développement X-Ray pour Java](#)
- [Kit de développement X-Ray pour Node.js](#)
- [Le démon X-Ray](#)

Carte de traçage X-Ray et pages de détails des traces

Les sections suivantes peuvent vous aider si vous rencontrez des problèmes lors de l'utilisation de la carte de suivi X-Ray et de la page de détails du suivi :

Je ne vois pas tous mes CloudWatch journaux

La manière de configurer les journaux afin qu'ils apparaissent sur la carte de suivi et les pages de détails des traces de X-Ray dépend du service.

- Les journaux API Gateway apparaissent si la journalisation est activée dans API Gateway.

Tous les nœuds de carte de service ne prennent pas en charge l'affichage des journaux associés. Afficher les journaux pour les types de nœuds suivants :

- Contexte Lambda
- fonction Lambda
- Étape API Gateway
- Cluster Amazon ECS
- Instance Amazon ECS
- Amazon ECS service
- Tâche Amazon ECS

- Cluster Amazon EKS
- Espace de noms Amazon EKS
- Nœud Amazon EKS
- Capsule Amazon EKS
- Service Amazon EKS

Je ne vois pas toutes mes alarmes sur la carte X-Ray Trace

La carte de suivi X-Ray affiche uniquement l'icône d'alerte d'un nœud si les alarmes associées à ce nœud sont en état ALARM.

La carte de trace associe les alarmes aux nœuds selon la logique suivante :

- Si le nœud représente un AWS service, toutes les alarmes associées à l'espace de noms associé à ce service sont associées au nœud. Par exemple, un nœud de type `AWS::Kinesis` est lié à toutes les alarmes basées sur les métriques de l'espace de CloudWatch nomsAWS/Kinesis.
- Si le nœud représente une AWS ressource, les alarmes relatives à cette ressource spécifique sont liées. Par exemple, un nœud du type `AWS::DynamoDB::Table` « MyTable » est lié à toutes les alarmes basées sur une métrique dont l'espace AWS/DynamoDB de noms est `TableName` défini sur `MyTable`.
- Si le nœud est de type inconnu, identifié par une bordure en pointillés autour du nom, aucune alerte n'est associée à ce nœud.

Je ne vois pas certaines AWS ressources sur la carte de suivi

Toutes les AWS ressources ne sont pas représentées par un nœud dédié. Certains AWS services sont représentés par un seul nœud pour toutes les demandes adressées au service. Les types de ressources suivants sont affichés avec un nœud par ressource :

- `AWS::DynamoDB::Table`
- `AWS::Lambda::Function`

Les fonctions Lambda sont représentées par deux nœuds : l'un pour le conteneur Lambda et l'autre pour la fonction. Cela aide à identifier les problèmes de démarrage à froid avec les fonctions Lambda. Les nœuds de conteneur Lambda sont associés à des alertes et des tableaux de bord de la même façon que les nœuds de fonction Lambda.

- `AWS::ApiGateway::Stage`
- `AWS::SQS::Queue`
- `AWS::SNS::Topic`

Il y a trop de nœuds sur la carte de trace

Utilisez des groupes X-Ray pour diviser votre carte en plusieurs cartes. Pour de plus amples informations, veuillez consulter [Utilisation des expressions de filtre avec des groupes](#).

Kit de développement X-Ray pour Java

Erreur : exception dans le thread « Thread-1" com.amazonaws.xray.exceptions.

SegmentNotFoundException: Impossible de commencer le sous-segment nommé « AmazonSNS » : le segment est introuvable.

Cette erreur indique que le SDK X-Ray a tenté d'enregistrer un appel sortant vers AWS, mais n'a pas trouvé de segment ouvert. Cela peut se produire dans les cas suivants :

- Aucun filtre de servlet n'est configuré : le SDK X-Ray crée des segments pour les demandes entrantes avec un filtre nommé. `AWSXRayServletFilter` [Configurez un filtre de servlet](#) pour instrumenter les demandes entrantes.
- Vous utilisez des clients instrumentés en dehors du code servlet. Si vous utilisez un client instrumenté pour passer des appels dans le code de démarrage ou dans un autre code qui ne s'exécute pas en réponse à une demande entrante, vous devez créer un segment manuellement. Pour obtenir des exemples, consultez [Instrumentation du code de démarrage](#).
- Vous utilisez des clients instrumentés dans les threads de travail : lorsque vous créez un nouveau thread, l'enregistreur X-Ray perd sa référence au segment ouvert. Vous pouvez utiliser les méthodes [getTraceEntity](#) et [setTraceEntity](#) pour obtenir une référence au segment ou sous-segment actuel ([Entity](#)) et le retransmettre à l'enregistreur à l'intérieur du thread. Veuillez consulter [Utilisation de clients instrumentés dans les threads de travail](#) pour obtenir un exemple.

Kit de développement X-Ray pour Node.js

Problème : CLS ne fonctionne pas avec Sequelize

Transmettez l'espace de noms du SDK X-Ray pour Node.js à Sequelize avec la méthode. `cls`

```
var AWSXRay = require('aws-xray-sdk');
const Sequelize = require('sequelize');
Sequelize.cls = AWSXRay.getNamespace();
const sequelize = new Sequelize(...);
```

Problème : CLS ne fonctionne pas avec Bluebird

Utilisez `cls-bluebird` pour que Bluebird fonctionne avec CLS.

```
var AWSXRay = require('aws-xray-sdk');
var Promise = require('bluebird');
var clsBluebird = require('cls-bluebird');
clsBluebird(AWSXRay.getNamespace());
```

Le démon X-Ray

Problème : Le démon utilise les mauvaises informations d'identification

Le démon utilise le AWS SDK pour charger les informations d'identification. Si vous utilisez plusieurs méthodes pour fournir les informations d'identification, la méthode avec la priorité la plus élevée est utilisée. Pour plus d'informations, consultez [Exécution du démon](#).

Historique du document pour AWS X-Ray

Le tableau suivant décrit les modifications importantes apportées à la documentation de AWS X-Ray. Pour recevoir les notifications de mise à jour de cette documentation, abonnez-vous à un flux RSS.

Dernière mise à jour de la documentation : 07 mars 2024

Modification	Description	Date
Fonctionnalité ajoutée	Migration de X-Ray vers. OpenTelemetry Pour plus d'informations, consultez la section Migration de l'instrumentation X-Ray vers l' OpenTelemetry instrumentation .	13 juin 2025
Fonctionnalité ajoutée	AWS X-Ray prend désormais en charge la recherche de transactions. Pour plus d'informations, consultez la section Recherche de transactions .	21 novembre 2024
Fonctionnalité ajoutée	AWS X-Ray prend désormais en charge OpenTelemetry le protocole OTLP (Protocol Endpoint). Pour de plus amples informations, veuillez consulter OpenTelemetry .	21 novembre 2024
Fonctionnalité ajoutée	X-Ray enregistre désormais les événements liés aux donnéesPutTraceSegments , notamment GetTraceSummaries , et BatchGetTraces à AWS	07 mars 2024

CloudTrail. X-Ray enregistre également désormais l'événement `GetSamplingStatisticSummaries` de gestion dans CloudTrail. Pour plus d'informations, consultez [Logging X-Ray API calls with AWS CloudTrail](#).

Fonctionnalité ajoutée

X-Ray prend désormais en charge les traces IDs créées via OpenTelemetry ou tout autre framework conforme à la spécification [W3C Trace Context](#). Pour plus d'informations, consultez la section [Envoi de données de suivi à X-Ray](#).

25 octobre 2023

Fonctionnalité ajoutée

Vous pouvez désormais configurer le suivi actif d'Amazon SNS, ce qui vous permet de suivre et d'analyser les demandes au fur et à mesure qu'elles passent par vos rubriques Amazon SNS. Pour plus d'informations, consultez [Amazon SNS et AWS X-Ray](#).

8 février 2023

Rubrique relative au SDK X-Ray pour Node.js mise à jour

Ajout de détails sur l'instrumentation des clients à l'aide de la AWS SDK pour JavaScript V3. Pour plus de détails, consultez la section [Tracing AWS SDK calls with the X-Ray SDK for Node.js](#).

7 février 2023

[Détails de la politique gérée par IAM mis à jour](#)

Ajout de l'autorisation IAM pour l'observabilité entre comptes AWSXRayFullAccess et AWSXrayCrossAccountSharingConfiguration politiques AWSXRayReadOnlyAccess gérées. Pour plus de détails, consultez les [politiques gérées par IAM pour X-Ray](#).

7 février 2023

[Fonctionnalité ajoutée](#)

AWS X-Ray prend désormais en charge l'observabilité entre comptes, vous permettant de surveiller et de dépanner les applications qui s'étendent sur plusieurs comptes au sein d'un. Région AWS Pour plus de détails, consultez la section [Suivi entre comptes](#).

27 novembre 2022

[Fonctionnalité ajoutée](#)

Vous pouvez désormais consulter les traces liées entre les producteurs de messages, une file d'attente Amazon SQS et les consommateurs, offrant ainsi une vue connectée des traces envoyées depuis des applications pilotées par des événements. Pour plus d'informations, consultez la section [Suivi des applications pilotées par des événements](#).

20 novembre 2022

[Détails de la politique gérée par IAM mis à jour](#)

Ajout de l'autorisation IAM pour répertorier les politiques de ressources dans la politique AWSXRayReadOnlyAccess gérée. Pour plus de détails, consultez les [politiques gérées par IAM pour X-Ray](#).

15 novembre 2022

[Permissions de console IAM mises à jour et détails des politiques gérées](#)

L'ensemble des autorisations IAM utilisées par la console X-Ray a été mis à jour, ainsi que la description de la politique AWSXRayReadOnlyAccess gérée. Pour plus de détails, consultez la section [Utilisation de la console X-Ray](#).

11 novembre 2022

[Ajout de AWS la distribution pour Ruby OpenTelemetry](#)

AWS Distro for OpenTelemetry (ADOT) fournit un ensemble unique d'open source APIs, de bibliothèques et d'agents pour collecter des traces et des métriques distribuées. ADOT Ruby vous permet d'instrumenter votre application Ruby pour X-Ray et d'autres backends de traçage. Pour plus d'informations, consultez [AWS Distro for OpenTelemetry Ruby](#).

7 février 2022

[Fonctionnalité ajoutée](#)

Vous pouvez désormais consulter les traces et configurer X-Ray depuis la CloudWatch console. Pour plus d'informations, consultez la section [X-Ray Console](#).

24 janvier 2022

[CloudWatch RUM intégré](#)

Avec AWS X-Ray et CloudWatch RUM, vous pouvez analyser et déboguer le chemin de la demande en commençant par les utilisateurs finaux de votre application via les services AWS gérés en aval. Pour plus d'informations, voir [CloudWatch RUM et AWS X-Ray](#).

3 décembre 2021

[AWS Distribution intégrée pour OpenTelemetry](#)

The AWS Distro for OpenTelemetry (ADOT) fournit un ensemble unique d'open source APIs, de bibliothèques et d'agents pour collecter des traces et des métriques distribuées. ADOT vous permet d'instrumenter votre application pour X-Ray et d'autres back-ends de suivi. Pour plus d'informations, consultez [Instrumentation de votre application](#).

23 septembre 2021

Fonctionnalité ajoutée	AWS X-Ray s'intègre désormais à Amazon Virtual Private Cloud, permettant aux ressources de votre Amazon VPC de communiquer avec le service X-Ray sans passer par l'Internet public. Pour plus d'informations, consultez la section Utilisation AWS X-Ray avec les points de terminaison VPC .	20 mai 2021
Fonctionnalité ajoutée	AWS X-Ray s'intègre désormais à X-Ray AWS CloudFormation, ce qui vous permet de provisionner et de configurer les ressources de X-Ray. Pour plus d'informations, consultez Creating X-Ray Resources with CloudFormation .	6 mai 2021
Fonctionnalité ajoutée	AWS X-Ray s'intègre désormais EventBridge à Amazon pour suivre les événements transmis EventBridge. Cela permet aux utilisateurs d'avoir une vue plus complète de leur système. Pour plus d'informations, consultez Amazon EventBridge et AWS X-Ray .	2 mars 2021

Ajout d'un daemon à ECR	Le daemon peut désormais être téléchargé depuis Amazon ECR. Pour plus d'informations, consultez la section Téléchargement du daemon .	1er mars 2021
Fonctionnalité ajoutée	AWS X-Ray prend désormais en charge les notifications relatives aux informations envoyées à Amazon EventBridge. Cela vous permet de prendre des mesures automatiques en fonction des informations que vous utilisez EventBridge. Pour plus d'informations, consultez la section Notifications Insights .	15 octobre 2020
Démonstrations téléchargeables ajoutées	AWS X-Ray introduit le démon de support pour Linux. ARM64 Pour plus d'informations, consultez AWS X-Ray daemonbrazil ws	1er octobre 2020

Fonctionnalité ajoutée

AWS X-Ray prend désormais en charge l'intégration active avec Amazon CloudWatch Synthetics. Cela vous permet de voir les détails d'un nœud client Synthetics Canary, tels que le temps de réponse et l'état. Vous pouvez également effectuer une analyse dans la console Analytics en fonction des informations provenant d'un nœud client Synthetics Canary. Pour plus d'informations, consultez la section [Débogage de canaris CloudWatch synthétiques](#) à l'aide de X-Ray.

24 septembre 2020

Fonctionnalité ajoutée

AWS X-Ray prend désormais en charge le suivi end-to-end des flux de travail pour AWS Step Functions. Vous pouvez visualiser les composants de votre machine d'état, identifier les goulots d'étranglement liés aux performances et résoudre les demandes qui ont entraîné une erreur. Pour plus d'informations, reportez-vous [AWS Step Functions aux sections et AWS X-Ray](#).

14 septembre 2020

Fonctionnalité ajoutée

AWS X-Ray fournit des informations permettant d'analyser en permanence les données de suivi de votre compte afin d'identifier les problèmes émergents dans vos applications. Insights enregistre les incidents et suit leur impact jusqu'à leur résolution. Pour plus d'informations, voir [Utilisation des informations dans la AWS X-Ray console](#)

3 septembre 2020

Fonctionnalité ajoutée

AWS X-Ray introduit l'agent d'auto-instrumentation Java, qui permet aux clients de collecter des données de suivi sans avoir à modifier l'application Java existante. Vous pouvez désormais suivre les applications Web et basées sur des servlets Java avec un minimum de modification de configuration et aucune modification de code. Pour plus d'informations, consultez la section [Agent AWS X-Ray d'instrumentation automatique pour Java](#).

3 septembre 2020

Fonctionnalité ajoutée

AWS X-Ray a ajouté une nouvelle page Groupes à la console X-Ray pour faciliter la création et la gestion de groupes de traces. Pour plus d'informations, consultez [Configuration de groupes dans la console X-Ray](#).

24 août 2020

Fonctionnalité ajoutée

AWS X-Ray permet désormais d'ajouter des balises aux groupes et aux règles d'échantillonnage. Vous pouvez également contrôler l'accès aux groupes et aux règles d'échantillonnage en fonction des balises. Pour plus d'informations, consultez les sections [Marquage des règles et des groupes d'échantillonnage par rayons X](#) et [Gestion de l'accès aux groupes de rayons X et aux règles d'échantillonnage basées sur les balises](#).

24 août 2020

Les traductions sont fournies par des outils de traduction automatique. En cas de conflit entre le contenu d'une traduction et celui de la version originale en anglais, la version anglaise prévaudra.