aws

開発者ガイド

AWS X-Ray

AWS X-Ray: 開発者ガイド

Table of Contents

とは AWS X-Ray

AWS X-Ray は、計測されたアプリケーションが行う受信レスポンスと呼び出しに関するトレース情報を提供します。これには、以下が含まれます。

- ダウンストリーム AWS リソース
- マイクロサービス
- データベース
- ウェブ APIs

トレースデータとビジュアライゼーションを使用して、アプリケーションのパフォーマンスに関するインサイトを取得し、問題を特定し、最適化の機会を見つけます。X-Ray の分析ツールを使用して、アプリケーションへのトレースされたリクエストの詳細を表示、フィルタリング、調査します。

X-Ray の仕組み

X-Ray を使用するには、X-Ray がアプリケーションがリクエストをどのように処理するかを追跡できるように、まずアプリケーションを計測する必要があります。アプリケーションにインストルメンテーションを追加すると、X-Ray はアプリケーション内の受信およびアウトバウンドリクエストやその他のイベントのトレースデータとメタデータを送信できます。例えば、Java アプリケーション AWS のサービス が行うすべての受信 HTTP リクエストと へのダウンストリーム呼び出しを計測できます。アプリケーションを自動的に計測することもできます。詳細については、「アプリケーションの計測」を参照してください。

X-Ray は、計測されたアプリケーションが受け取るすべてのリクエストにトレース ID を割り当てます。アプリケーションが別のコンポーネントとやり取りする場合、X-Ray はセグメントを作成します。このセグメントは元のトレース ID に関連付けられ、そのコンポーネントとのやり取りの品質を追跡します。

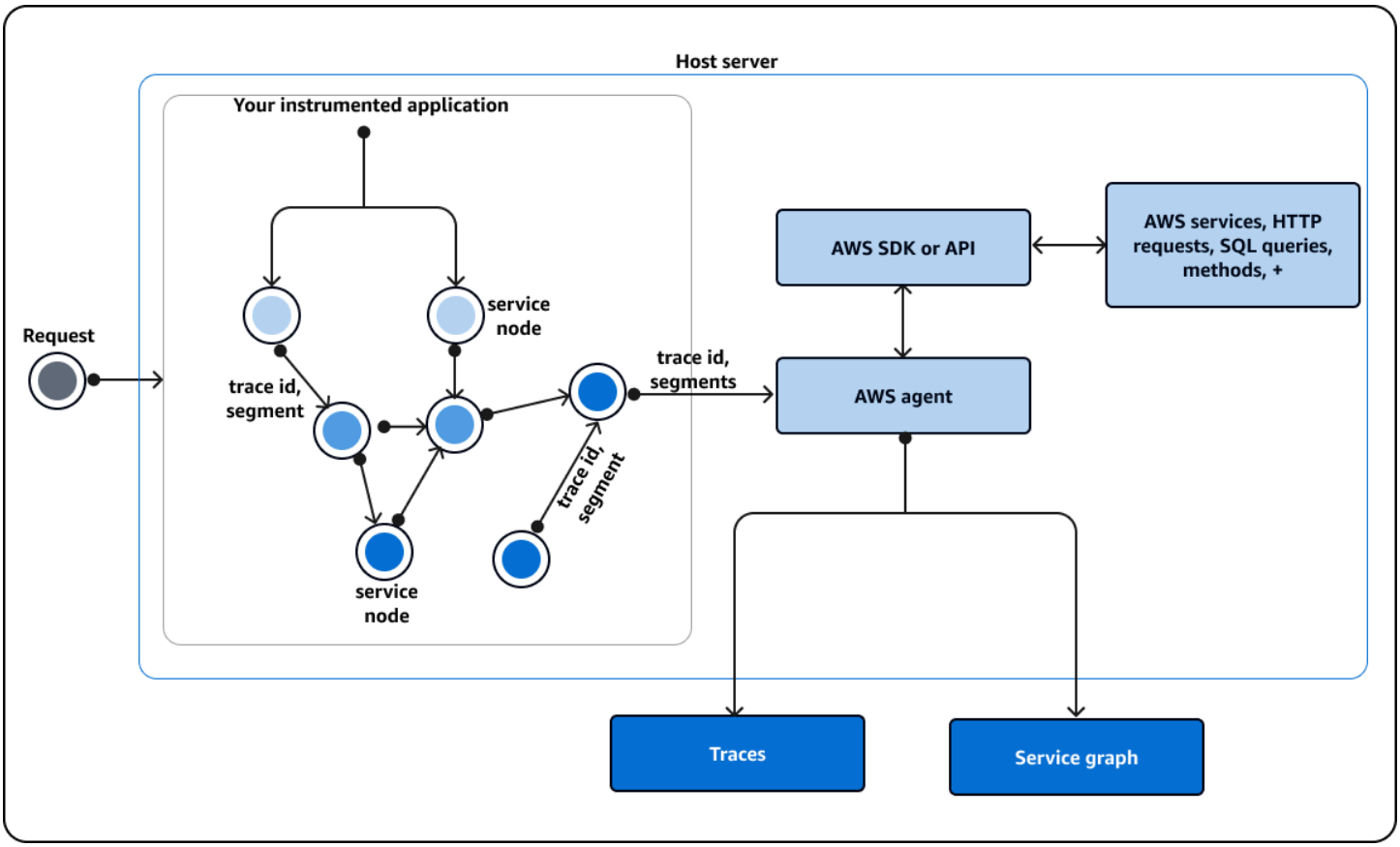
X-Ray は、アプリケーションワークフロー全体でトレース ID とセグメントを追跡します。ワークフロー全体を分析したり、詳細分析のためにピースを分離したりできます。セグメントの詳細については、次の概念セクションを参照してください。

X-Ray は、アプリケーションがサービスノードまたはコンポーネントとやり取りするときに追跡し、受信リクエストを次のように処理します。

1. X-Ray はトレース ID とセグメントを使用して個々のインタラクションを追跡します。

2. AWS エージェントはトレース ID と関連するセグメントを収集し、SDK または API トレースフレームワークに渡します。
3. X-Ray は、X-Ray と統合されている AWS サービスとのインタラクションも追跡します。
4. エージェントはデータをコンソール GUI に送信し、トレース、セグメント、サブセグメント、およびこれらのコンポーネントの相互作用に関する情報を表示できます。

前のステップを次の図に示します。



X-Ray がインストルメント化されたアプリケーションとやり取りする方法

計測されたアプリケーションがリクエストを受信すると、X-Ray は以下を実行します。

1. アプリケーションがリクエストを処理すると、X-Ray SDK はトレースデータを AWS コレクターまたはエージェント に送信します。次に、エージェントはトレース ID とセグメントを収集します。次の 3 つのエージェントから選択できます。

   - AWS Distro for OpenTelemetry (ADOT) Collector – オープンソースの標準化されたOpenTelemetryエージェント に基づいて AWS、 によって最適化および保護されるオープン

ソースコレクター。言語やベンダーに依存しない標準化されたコードADOT Collectorを使用してエージェントとやり取りするが、 AWS セキュリティと最適化の信頼度がエンドユーザーに組み込まれている場合は、 を使用します。ADOT を使用して、エンドポイントをさまざまなエージェントやバックエンドに設定することもできます。

- Amazon CloudWatch エージェント – ログ、メトリクス、トレースを統合し、すべてのテレメトリデータをサポートし、 をそのデータADOT Collectorに統合したオープンソースコレクター。
- X-Ray デーモン — X-Ray SDK および X-Ray APIs レガシーコードがある場合、またはカスタマイズされたトレースを必要とするアプリケーションがあり、X-Ray APIsを使用する必要がある場合は、X-Ray デーモンを使用します。デーモンは、Linux、、Microsoft Windowsおよびで利用できmacOS、 AWS Elastic Beanstalk および AWS Lambda プラットフォームに含まれています。

2. 次に、エージェントは API または AWS API 上に構築された SDK のいずれかで構成されるトレースフレームワークにこのデータを送信します。 AWS このフレームワークは、他の AWS サービスとやり取りします。X-Ray API は、 AWS SDK を介して AWS Command Line Interface、または 経由で直接、すべての X-Ray 機能へのアクセスを提供しますHTTPS。言語を使用している場合、または SDK でサポートされていないオペレーションが必要な場合は、X-Ray API を使用します。

   次の SDKsを使用できます。

   - ADOT SDK – ADOT SDK を使用して、 に関連しないベンダーのさまざまなエージェントとやり取りします AWS。ADOT SDK は複数のバックエンドサービスもサポートしています。
   - X-Ray SDK – 機能や言語を追加しなくなったクラシック製品。アプリケーションコードを更新しない場合は、X-Ray SDK を使用します。

   X-Ray または ADOT SDK を使用している場合は、 エージェントと組み合わせて次のオプションを使用できます。

   - CloudWatch エージェントで X-Ray または ADOT SDK を使用する – 推奨。
   - で ADOT SDK を使用する ADOT Collector – セキュリティと最適化の AWS レイヤーでベンダーに依存しないソフトウェアを使用する場合は、 が推奨されます。
   - CloudWatch エージェントで X-Ray SDK を使用する – CloudWatch エージェントは X-Ray SDK と互換性があります。
   - X-Ray SDK を X-Ray デーモンで使用する – X-Ray SDK を引き続き使用する場合に使用します。

3. （オプション) トレースフレームワークは、他の AWS サービス、HTTPサーバー、その他のメソッドやクエリとやり取りできます。X-Ray と統合する一部の AWS サービスには、Amazon EC2 Amazon SNS と API Gateway が含まれます。SDK または API は、これらのインタラクション中のトレースデータを追跡します。

   AWS X-Ray と統合する のサービスは、受信リクエストにトレースヘッダーを追加したり、X-Ray にトレースデータを送信したり、エージェントを実行してトレースデータを収集したりできます。例えば、 AWS Lambda はリクエストに関するトレースデータを Lambda 関数に送信できます。

   X-Ray と連携する他のサービスの詳細については、「」を参照してください他の AWS X-Ray との統合 AWS のサービス。

4. トレース、セグメント、およびサブセグメントに関するデータは、グラフィカルユーザーインターフェイス (GUI) でコンソールで表示できます。次のオプションを設定できます。

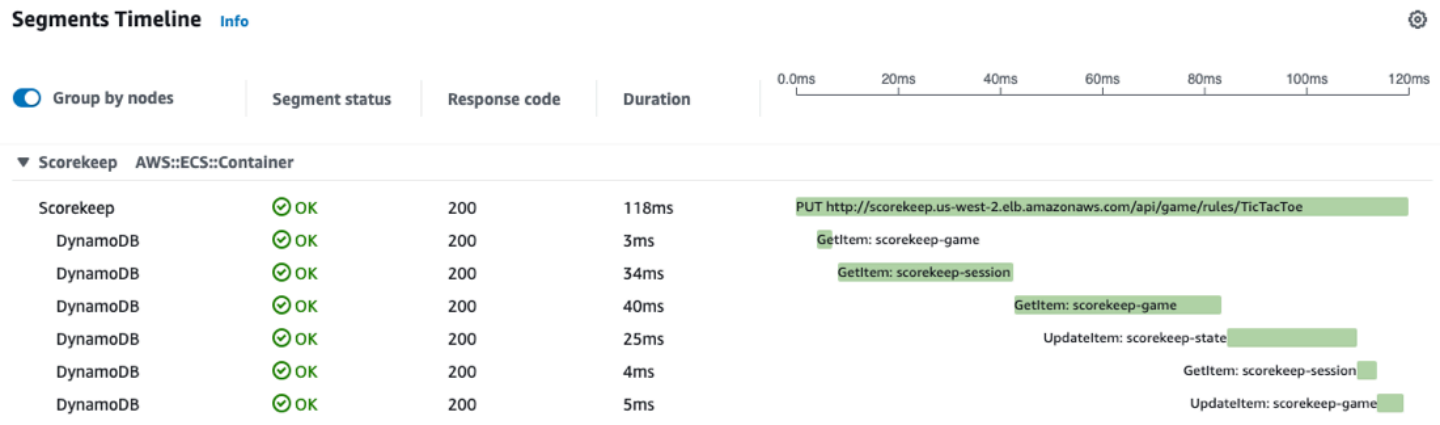
   - https://console.aws.amazon.com/cloudwatch/ – トレース、ログ、メトリクスを 1 か所で表示するための GUI エクスペリエンス。X-Ray サービスマップとレガシー CloudWatchServiceLensマップは、 CloudWatch コンソール内の X-Ray トレースマップに結合されます。
   - https://console.aws.amazon.com/xray/home — トレースに関する情報を表示できる GUI エクスペリエンス。トレース、トレースマップ、サービスマップ、分析に関するインサイトを含む情報を表示できます。 AWS は、このコンソールエクスペリエンスを開発していません。

X-Ray は、アプリケーションが操作する AWS リソースからのトレースデータを使用して、詳細なトレースマップ を生成します。トレースマップには、フロントエンドサービスが 1 回のリクエストで呼び出すクライアント、フロントエンドサービス、バックエンドサービスが表示されます。トレースマップを使用して、アイデンティティのボトルネック、レイテンシースパイク、その他の問題を特定し、アプリケーションのパフォーマンスを解決または改善します。

X-Ray は、アプリケーションがサービスノードとやり取りする方法の全体像を示すサービスマップも生成します。サービスマップのエッジは、サービスノードを接続します。ノードが相互に通信する頻度と、それらの通信のレイテンシーを示します。

次の図は、アプリケーションが異なるコンポーネントとどのように相互作用するかを示すサービスマップの例を示しています。コンソールでサービスマップを表示できます。イメージは、クライアントからリクエストを受信するアプリケーションを示しています。次に、このイメージは、アプリケーションが 2 つの DynamoDB テーブルと Amazon SNS とどのようにやり取りするかを示しています。

次の画像は、トレース内の 1 つのセグメントのコンソールで使用できるデータの例です。この図は、複数のセグメントと、各セグメントが他のセグメントに関連して実行された開始時間と期間を一覧表示したタイムラインを示しています。このイメージには、セグメントのステータスと HTTP レスポンスコードも表示されます。



概念

AWS X-Ray は、 サービスからデータをセグメント として受け取ります。X-Ray は、トレースへの共通リクエストを含むセグメントをグループ化します。X-Ray は、トレースを処理して、アプリケーションのビジュアル表現を提供するサービスグラフを生成します。

概念

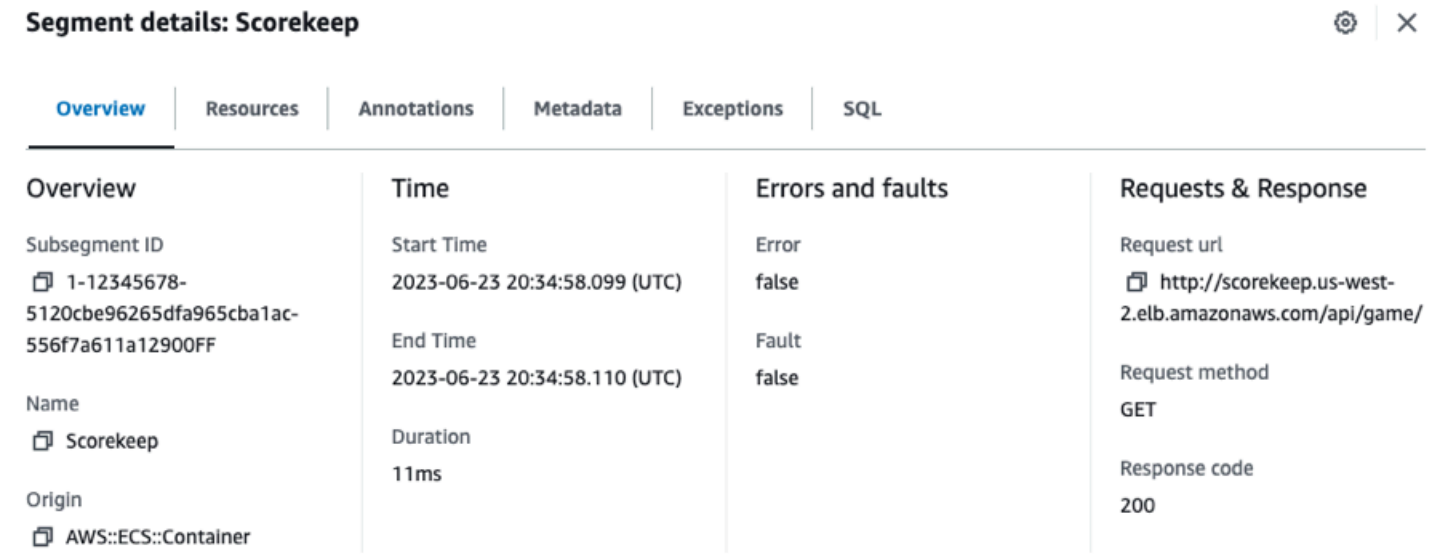
- セグメント
- サブセグメント
- サービスグラフ
- トレース
- サンプリング
- トレースヘッダー
- フィルタ式
- グループ
- 注釈とメタデータ
- エラー、障害、および例外

セグメント

アプリケーションロジックを実行しているコンピューティングリソースは、セグメントとしての動作に関するデータを送信します。セグメントには、リソース名、リクエストの詳細、行った作業の詳細が含まれています。たとえば、HTTP リクエストがアプリケーションに到達すると、次のデータが記録されます。

- ホスト – ホスト名、エイリアス、または IP アドレス。
- リクエスト – メソッド、クライアントアドレス、パス、ユーザーエージェント。
- レスポンス – ステータス、コンテンツ。
- 完了した作業 — 開始時刻と終了時刻、サブセグメント。
- 発生する問題 – エラー、障害、例外 (例外スタックの自動取得を含む)。

次の画像は、セグメントについて返される概要情報の例です。イメージには、ID、開始時刻と終了時刻、エラーまたは障害、HTTP リクエストからのリクエストとレスポンスコードに関する情報が表示されます。



SDKs または APIsで構成されるトレースフレームワークは、リクエストヘッダーとレスポンスヘッダー、アプリケーション内のコード、およびアプリケーションが実行される AWS リソースに関するメタデータから情報を収集します。X-Ray が収集するデータを選択するには、アプリケーション設定またはコードを変更して、受信リクエスト、ダウンストリームリクエスト、および AWS サービスを計測します。

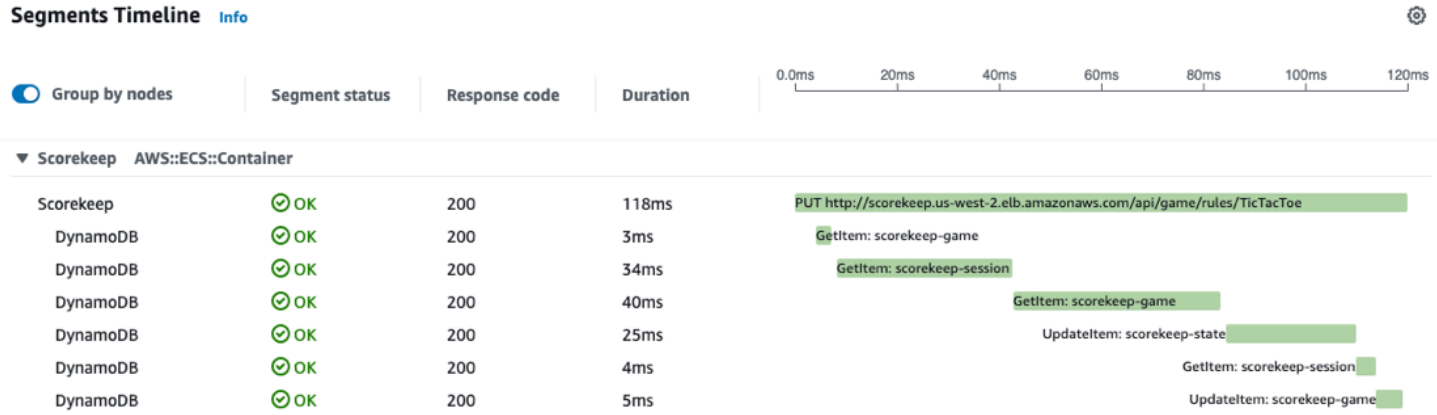
> **転送されたリクエスト**
>
> ロードバランサーまたは他の仲介者がアプリケーションにリクエストを転送する場合、X-Ray は、クライアントの IP をIP パケットの送信元 IP からではなく、リクエストのX-Forwarded-Forヘッダーから取得します。転送されたリクエストについて記録されたクライアント IP は偽造される可能性があるため、信頼されるべきではありません。

SDK や API などのトレースフレームワークを使用して、注釈やメタデータなどの詳細情報を記録できます。セグメントとサブセグメントの構造と記録された情報の詳細については、「」を参照してくださいX-Ray セグメントドキュメント。セグメントのドキュメントのサイズは最大 64 kB まで可能です。

サブセグメント

セグメントをサブセグメント に分割できます。サブセグメントは、アプリケーションが元のリクエストを処理するために行うダウンストリーム呼び出しに関するより詳細なタイミング情報と詳細を提供します。サブセグメントには、、外部 HTTP API AWS のサービス、または SQL データベースへ

の呼び出しに関する追加の詳細が含まれています。サブセグメントを定義して、アプリケーション内の特定の関数またはコード行を計測することもできます。



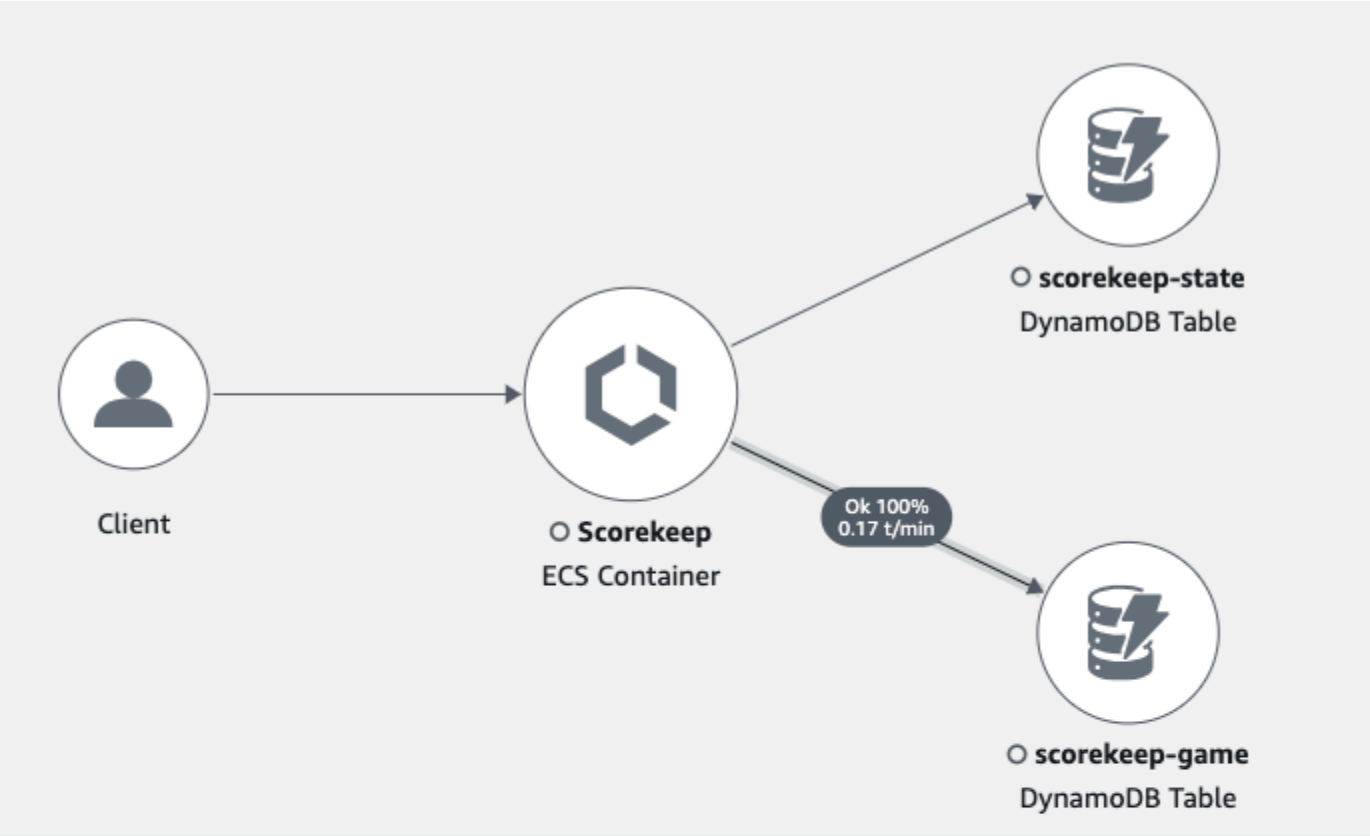
X-Ray はサブセグメントを使用して、Amazon DynamoDB など、独自のセグメントを送信しないサービスのトレースマップ上に推定セグメントとダウンストリームノードを生成します。サブセグメントを使用すると、依存関係がトレースをサポートしていない場合や、 の外部にある場合でも、ダウンストリームのすべての依存関係を確認できます AWS。

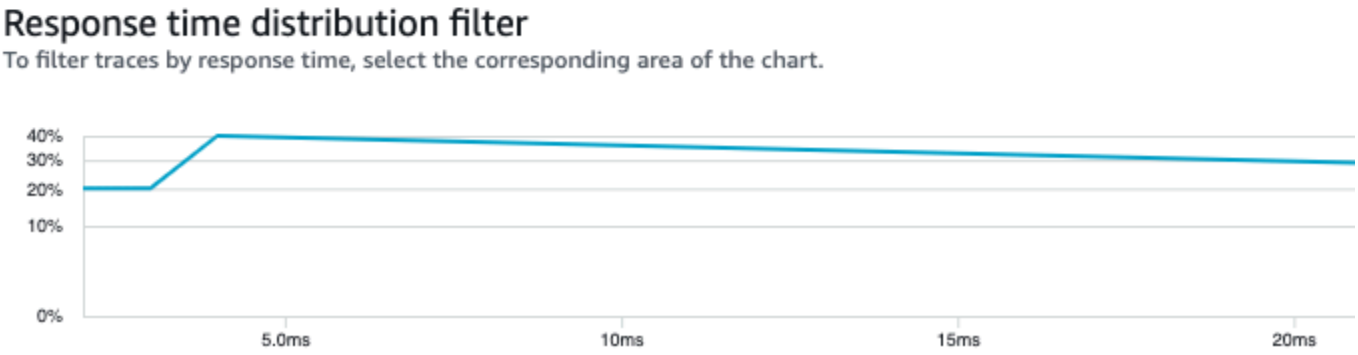
サブセグメントは、クライアントとしてのダウンストリーム呼び出しのアプリケーションビューを表します。ダウンストリームサービスも計測されている場合、そのセグメントはアップストリームクライアントのサブセグメントから推測されたセグメントを置き換えます。サービスグラフ上のノードは、利用可能な場合、サービスのセグメントからの情報を使用します。2 つのノード間の エッジは、アップストリームサービスのサブセグメントを使用します。

例えば、計測された AWS SDK クライアントで DynamoDB を呼び出すと、X-Ray SDK はその呼び出しのサブセグメントを記録します。DynamoDB はセグメントを送信しないため、サブセグメントには以下に関する情報が含まれます。

- トレース内の推定セグメント。
- サービスグラフの DynamoDB ; ノード。
- サービスと DynamoDB 間の エッジ。

次の図は、サンプルアプリケーションのサービスマップを示しています。イメージでは、クライアントはサンプルの Scorekeep アプリケーションにリクエストを行います。Scorekeep アプリケーションは DynamoDB に 2 つの呼び出しを行います。サービスマップの エッジは、これらの各呼び出しを表します。エッジを選択すると、DynamoDB テーブルに対して行われた呼び出しのヘルスステータス、数、および頻度が表示されます。次の画像は、レスポンスタイムでフィルタリングされた エッジに対応するトレースを示しています。

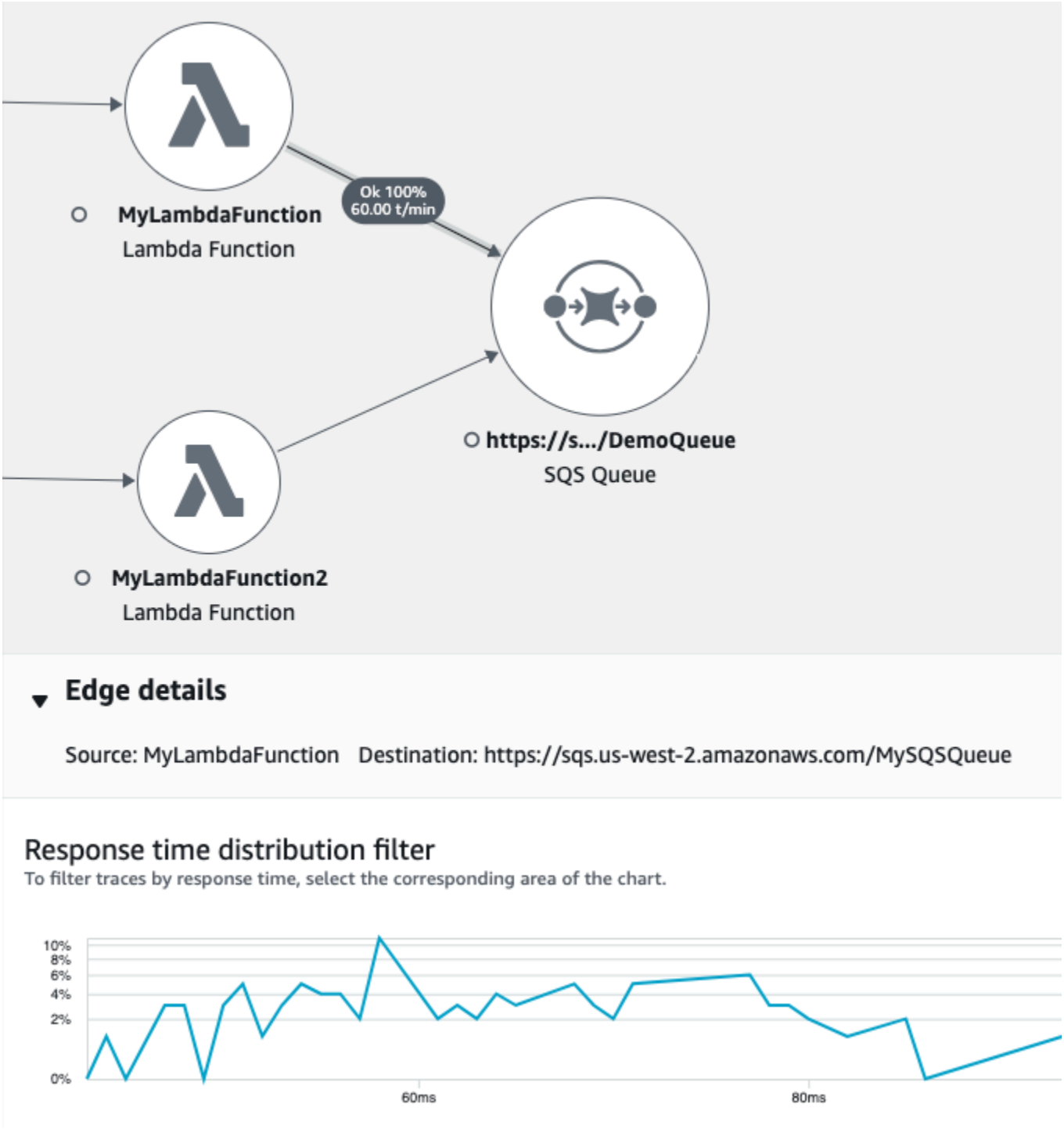




計測されたアプリケーションで別の計測されたサービスを呼び出すと、ダウンストリームサービスは独自のセグメントを送信します。このセグメントは、アップストリームサービスがサブセグメントに記録したのと同じ呼び出しのビューを記録します。サービスグラフでは、両方のサービスのノードにセグメントからのタイミングとエラー情報が含まれています。それらの間のエッジには、アップストリームサービスのサブセグメントからの情報が含まれます。ダウンストリームサービスは、リクエ

ストの作業を開始および終了した時刻を記録します。アップストリームサービスは、リクエストが 2 つのサービス間の移動に費やした時間を含むラウンドトリップレイテンシーを記録します。

次の図は、アップストリーム Lambda 関数に対応するエッジからの応答時間でフィルタリングされたトレース情報を示しています。



サービスグラフ

X-Ray はアプリケーションが送信したデータを使用してサービスグラフを生成します。X-Ray にデータを送信する各 AWS リソースは、グラフにサービスノードとして表示されます。エッジは、リクエストを処理するために連携するサービスを接続し、クライアントをアプリケーションに接続し、アプリケーションが使用するダウンストリームサービスとリソースに接続します。

> **サービス名**
>
> セグメントの `name` は、セグメントを生成するサービスのドメイン名または論理名と一致する必要があります。ただし、これは強制ではありません。`PutTraceSegments`にアクセス許可を持つアプリケーションは、任意の名前でセグメントを送信できます。

サービスグラフは、アプリケーションを構成するサービスおよびリソースに関する情報を含む JSON ドキュメントです。X-Ray コンソールでは、サービスグラフを使用して、視覚化またはサービスマップを生成します。

次の図は、サービスマップを示しています。サービスマップには、アプリケーションに対するクライアントのリクエストと、アプリケーションがリクエストを処理するために操作するサービスとの関係が表示されます。次の画像では、サンプル Scorekeep アプリケーションが 2 つの DynamoDB テーブルと Amazon SNS とやり取りしています。

分散アプリケーションでは、X-Ray は、同じトレース ID でリクエストを処理するすべてのサービスのノードを 1 つのサービスグラフに結合します。リクエストがインタラクションする最初のサービスには、フロントエンドと呼び出すサービスの間で伝達されるトレースヘッダーが追加されます。

例えば、Scorekeep は AWS Lambda 関数を呼び出してランダムな名前を生成するウェブ API を実行します。次に、X-Ray SDK はトレース ID を生成し、Lambda 関数への呼び出しを追跡します。AWS Lambda トレースデータとトレース ID を Lambda 関数に渡します。X-Ray SDK は、同じトレース ID を使用してエージェントまたはコレクターにデータを送信します。その結果、API、 AWS Lambda サービス、および Lambda 関数のノードはすべて、トレースマップ上に個別の接続ノードとして表示されます。

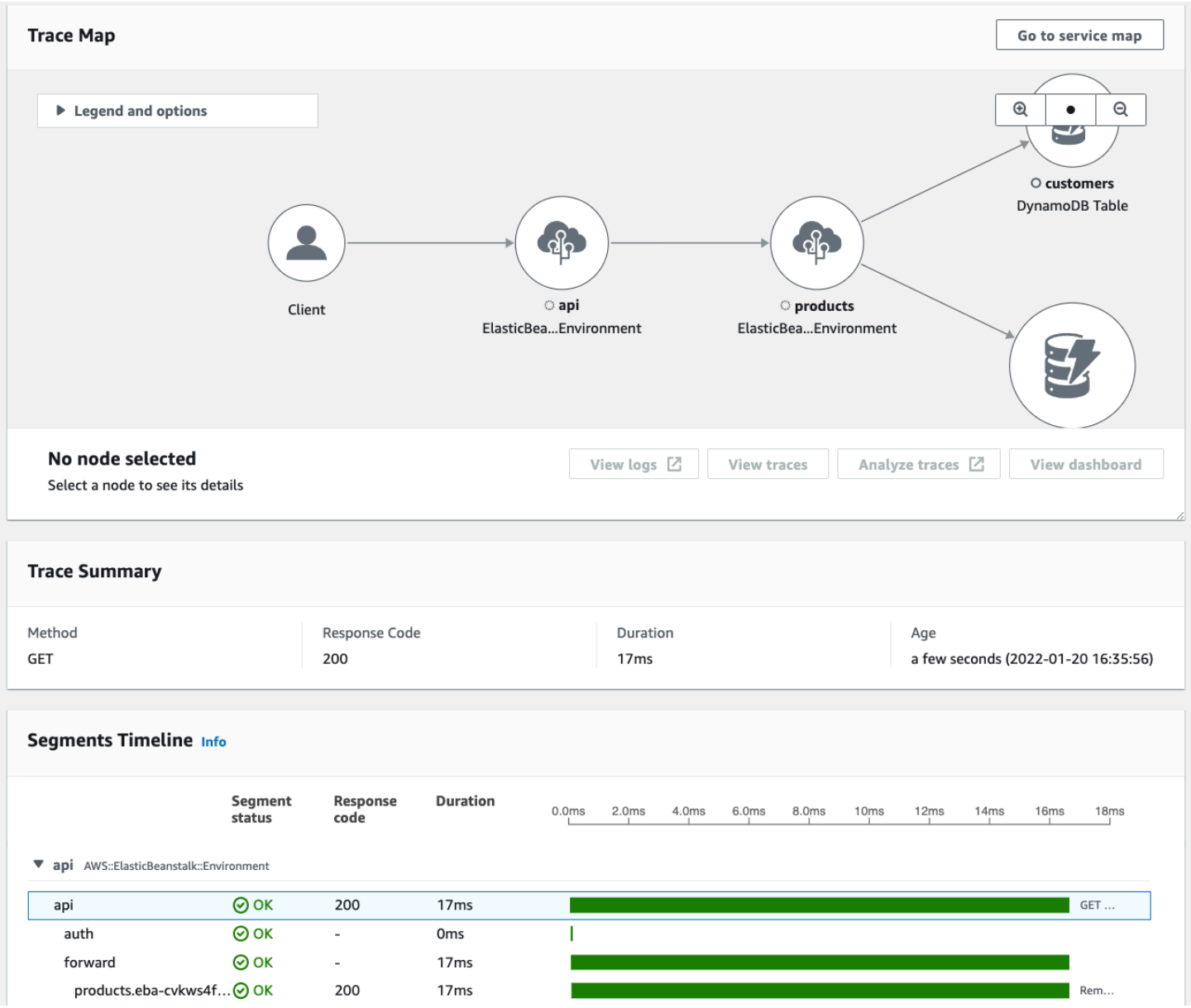
サービスグラフのデータは 30 日間保持されます。

トレース

トレースでは、1 つのリクエストで生成されたセグメントをすべて収集します。トレースはトレース ID を使用して、アプリケーション経由のリクエストのパスを追跡します。このリクエストは通常、

ロードバランサーを経由し、アプリケーションコードとやり取りし、他の AWS のサービスまたは外部ウェブ APIs。HTTP リクエストがインタラクションする最初のサポート対象サービスは、トレース ID ヘッダーをリクエストに追加します。ITheサービスはトレース ID をダウンストリームに伝播して、レイテンシー、処理、およびその他のリクエストデータを追跡します。

次の図は、 HTTPリクエストを処理するアプリケーションの例を示しています。トレースの概要には、HTTPレスポンスコード、リクエストの処理時間、アプリケーションがリクエストを処理した時間が含まれます。次の図は、各トレースセグメントのタイムラインも示しています。タイムラインには、ステータス、HTTPレスポンスコード、セグメントの終了にかかった時間が表示されます。グラフには、トレース内の各セグメントの他のセグメントに対する期間、開始時刻、終了時刻が表示されます。



X-Ray がトレースコレクションに請求する方法の詳細については、「X-Ray トレースの請求方法に関するAWS X-Ray 料金」を参照してください。トレースデータは 30 日間保持されます。

サンプリング

X-Ray SDK はサンプリングアルゴリズムを適用して、効率的なトレースを確保し、アプリケーションが処理するリクエストの代表的なサンプルを提供します。このアルゴリズムは、トレースするリクエストを決定します。デフォルトでは、X-Ray SDK は 1 秒ごとに最初に受信したリクエストと、追加のリクエストの 5% を記録します。

開始時にサービス料がかからないように、デフォルトのサンプリングレートは控えめになっています。デフォルトのサンプリングレートを変更するように X-Ray を設定し、サービスまたはリクエストのプロパティに基づいてサンプリングを適用する追加のルールを設定できます。

例えば、サンプリングを無効にして、状態を変更したり、ユーザーやトランザクションを処理したりする呼び出しのすべての要求をトレースすることができます。バックグラウンドポーリング、ヘルスチェック、接続メンテナンスなど、大量の読み取り専用呼び出しの場合。

詳細については、サンプリングルールを設定する「」およびCreateSamplingRule「 API」を参照してください。

トレースヘッダー

すべてのリクエストは、設定できる最小数までトレースされます。その最小値に達した後、X-Ray は追加コストを回避するためにリクエストのごく一部のみをトレースします。X-Ray は、 で始まるトレースヘッダーの HTTP リクエストにサンプリング決定とトレース ID を追加しますX-Amzn-Trace-Id。X-Ray は、リクエストが X-Ray と統合されている最初の AWS サービスとやり取りするときに、これらのヘッダーを追加します。X-Ray SDK はこれらのヘッダーを読み取り、レスポンスに含めます。

Example ルートトレース ID 突きのトレースヘッダーおよびサンプリングデシジョン

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

> **トレースヘッダーのセキュリティ**
>
> トレースヘッダーは、X-Ray SDK、 AWS のサービス、またはクライアントリクエストから発信できます。アプリケーションで受信リクエストから X-Amzn-Trace-Id を削除し、

ユーザーが自分のリクエストにトレース ID またはサンプリングデシジョンを追加することで発生する問題を回避できます。

計測対象アプリケーションからのリクエストの場合は、親セグメント ID をトレースヘッダーに含めることもできます。たとえば、アプリケーションで計測対象 HTTP クライアントを使用してダウンストリーム HTTP ウェブ API を呼び出す場合、X-Ray SDK は元のリクエストのセグメントの ID をダウンストリームリクエストのトレースヘッダーに追加します。ダウンストリームリクエストを処理する計測アプリケーションは、親セグメント ID を使用して 2 つのリクエストを接続します。

Example ルートトレース ID、親セグメント ID およびサンプリングデシジョンを含むトレースヘッダー

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

Lambda やその他の は、処理メカニズムの一部として で始まるヘッダーLineageの一部を追加するAWS のサービス 場合があります。トレースヘッダーの付加部分を直接使用しないでください。

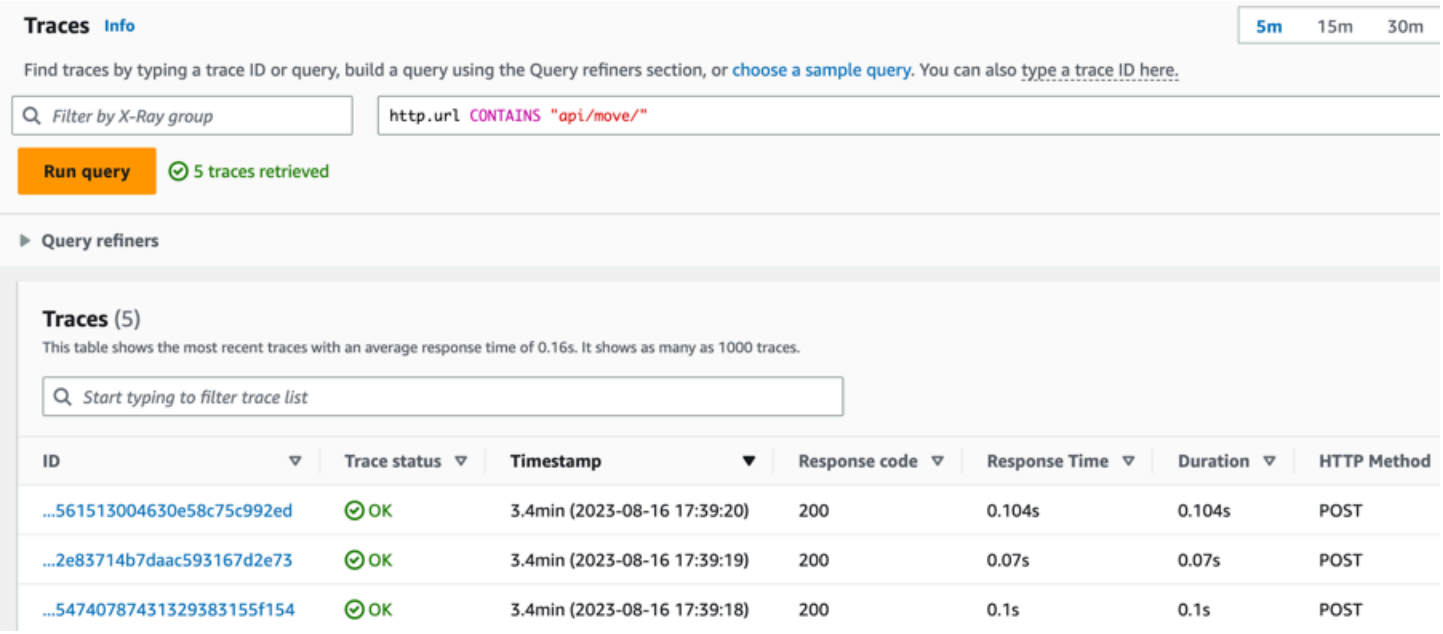
Example Lineage を含むトレースヘッダー

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

フィルタ式

データの小さなサブセットをサンプリングした場合でも、複雑なアプリケーションは大量のトレースデータを生成できます。フィルター式を使用すると、個々のリクエスト、特定のパス、ユーザーに関するトレースなど、特定のトレースを検索できます。

次の図は、定義したグループでフィルタリングするために使用できる X-Ray コンソールのテキストボックスを示しています。グループの詳細については、次の「グループ」セクションを参照してください。



グループ

フィルター式内のグループを使用して、トレースデータの量を減らし、グループ基準に適合するデータに集中できます。

グループを使用して、そのグループに固有のサービスグラフ、トレースの概要、 CloudWatch メトリクスを生成します。名前または Amazon リソースネーム (ARN) で を呼び出すことができます。X-Ray は、受信トレースを X-Ray サービスに保存されているグループフィルター式と照合します。CloudWatch は、グループ基準に一致するトレースのメトリクスを 1 分ごとに発行します。

グループのフィルタ式を更新しても、すでに記録されているデータは変わりません。更新は後続のトレースにのみ適用されます。これにより、新しい式と古い式がマージされたグラフが表示される場合があります。接続されていないグループが 1 つのグラフ内にマージされないようにするには、現在のグループを削除して新しいグループhttps://docs.aws.amazon.com/xray/latest/api/API_CreateGroup.htmlを作成します。

> **Note**
>
> グループの請求は、フィルター式に一致する取得トレースの数に基づきます。詳細については、「AWS X-Ray 料金表」を参照してください。

グループの詳細については、「グループを設定する」を参照してください。

注釈とメタデータ

アプリケーションを計測すると、X-Ray SDK は受信リクエストと送信リクエストに関する情報を記録します。SDK は、使用された AWS リソースとアプリケーション自体に関する情報も記録します。その他の情報を注釈およびメタデータとして、セグメントドキュメントに追加することもできます。注釈とメタデータはトレースレベルで結合されます。これらは、任意のセグメントまたはサブセグメントに追加できます。

アノテーションは、フィルター式 で使用するインデックスが作成されたキーと値のペアです。注釈を使用して、コンソールでトレースをグループ化するため、または`GetTraceSummaries` API を呼び出すときに使用するデータを記録します。

X-Ray は、トレースごとに 50 の注釈までインデックスを付けます。

メタデータは、オブジェクトやリストなど、インデックス化されていない任意のタイプの値を持つキーと値のペアです。メタデータを使用してトレースに保存するデータを記録しますが、トレースの検索用に使用する必要はありません。

セグメントまたはサブセグメントの詳細ウィンドウで、 CloudWatch コンソールのトレースの詳細ページ内で注釈とメタデータを表示できます。詳細については、「 でトレースとトレースの詳細を表示する」を参照してくださいX-Ray コンソールの詳細。

エラー、障害、および例外

X-Ray は、アプリケーションコードのエラーとダウンストリームサービスによって返されたエラーを追跡します。X-Ray は、リクエストから次のHTTPレスポンスコードを追跡します。

- **`Error`** – クライアントエラー (400 シリーズエラー) は、リクエスト自体にエラーが含まれているため、サーバーがクライアントからのリクエストを理解または処理できなかったことを示します。これらのエラーは、構文エラー、情報不足、または不正なリクエストボディが原因で発生する可能性があります。
- **`Fault`** – サーバー障害 (500 シリーズエラー) は、サーバー自体に問題があるため、サーバーが有効なリクエストを処理できなかったことを示します。これらのエラーは、ソフトウェアやハードウェアの障害、サーバーのリソース制限などの問題が原因で発生する可能性があります。
- **`Throttle`** – スロットリングエラー (429 リクエストが多すぎます) は、クライアントが一定期間にわたってサーバーまたは API に送信するリクエストが多すぎる場合に発生する特定のタイプのクライアントエラーです。

アプリケーションが計測されたリクエストを処理している間に例外が発生した場合、X-Ray SDK は、スタックトレース ID を含む例外の詳細を、可能な場合は記録します。X-Ray コンソールでセグメント詳細の例外を表示できます。

X-Ray の使用を開始する

X-Ray を使用するには、以下を実行する必要があります。

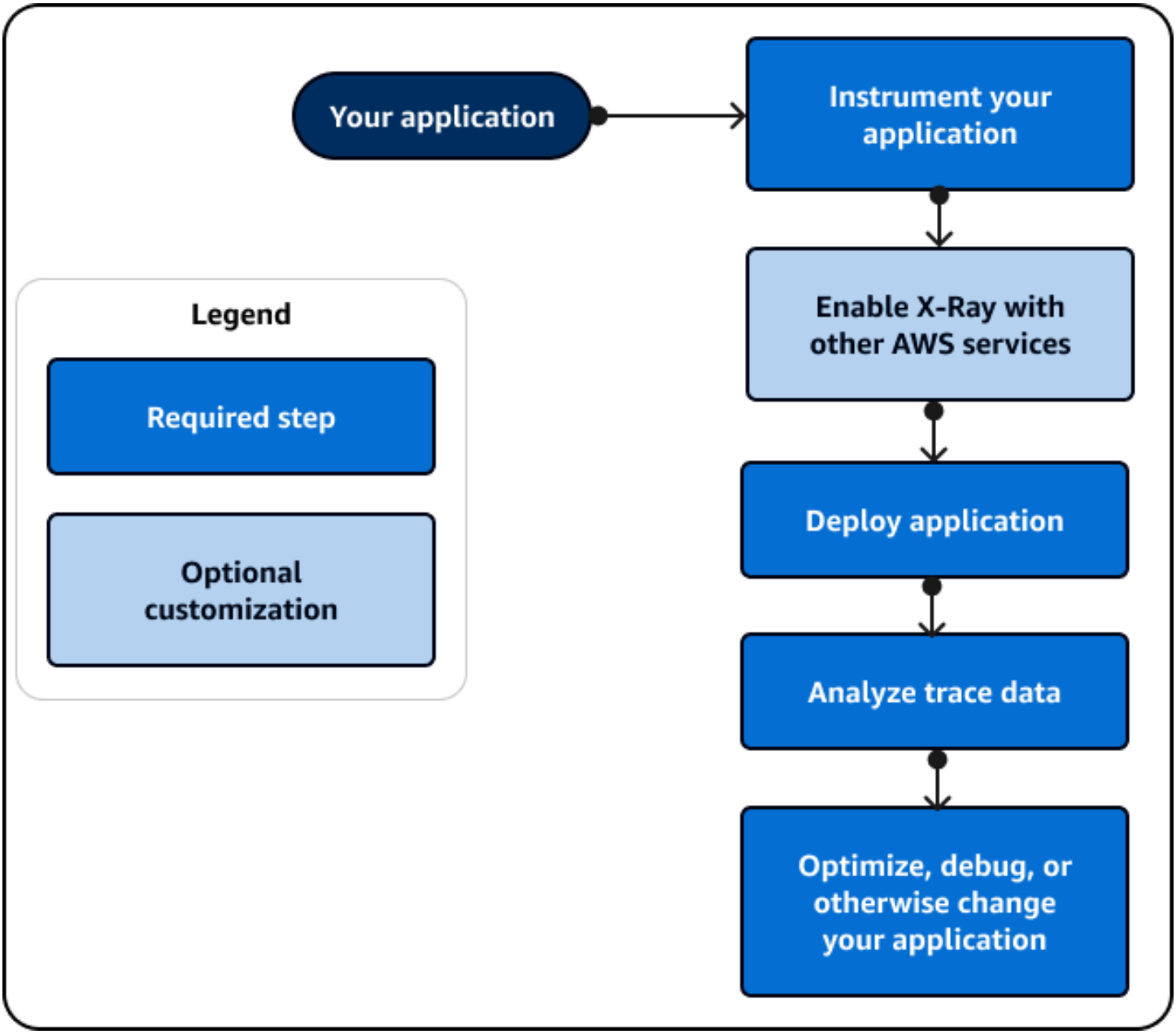
1. アプリケーションを計測します。これにより、X-Ray はアプリケーションがリクエストを処理する方法を追跡できます。

   - X-Ray SDKsX-Ray APIs、ADOTまたは CloudWatch Application Signals を使用して、X-Ray にトレースデータを送信します。使用するインターフェイスの詳細については、「」を参照してくださいインターフェイスを選択する。

   インストルメンテーションの詳細については、「」を参照してくださいのアプリケーションを計測する AWS X-Ray。
2. （オプション) X-Ray と統合 AWS のサービス する他の と連携するように X-Ray を設定します。トレースをサンプリングして受信リクエストにヘッダーを追加し、エージェントまたはコレクターを実行して、トレースデータを X-Ray に自動的に送信できます。詳細については、「他の AWS X-Ray との統合 AWS のサービス」を参照してください。
3. 実装したアプリケーションをデプロイします。アプリケーションがリクエストを受信すると、X-Ray SDK はトレース、セグメント、サブセグメントのデータを記録します。このステップでは、IAM ポリシーを設定し、エージェントまたはコレクターをデプロイする必要がある場合もあります。

   - AWS Distro for OpenTelemetry (ADOT) SDK と CloudWatch エージェントを使用してアプリケーションをさまざまなプラットフォームにデプロイするスクリプトの例については、「Application Signals Demo Scripts」を参照してください。
   - X-Ray SDK と X-Ray デーモンを使用してアプリケーションをデプロイするスクリプトの例については、「」を参照してくださいAWS X-Ray サンプルアプリケーション。
4. （オプション) コンソールを開いて、データを表示および分析します。トレースマップ、サービスマップなどの GUI 表現を表示して、アプリケーションがどのように機能するかを検査できます。コンソールでグラフィカル情報を使用して、アプリケーションを最適化、デバッグ、理解します。コンソールの選択の詳細については、「」を参照してくださいを使用する AWS Management Console。

次の図は、X-Ray の使用を開始する方法を示しています。



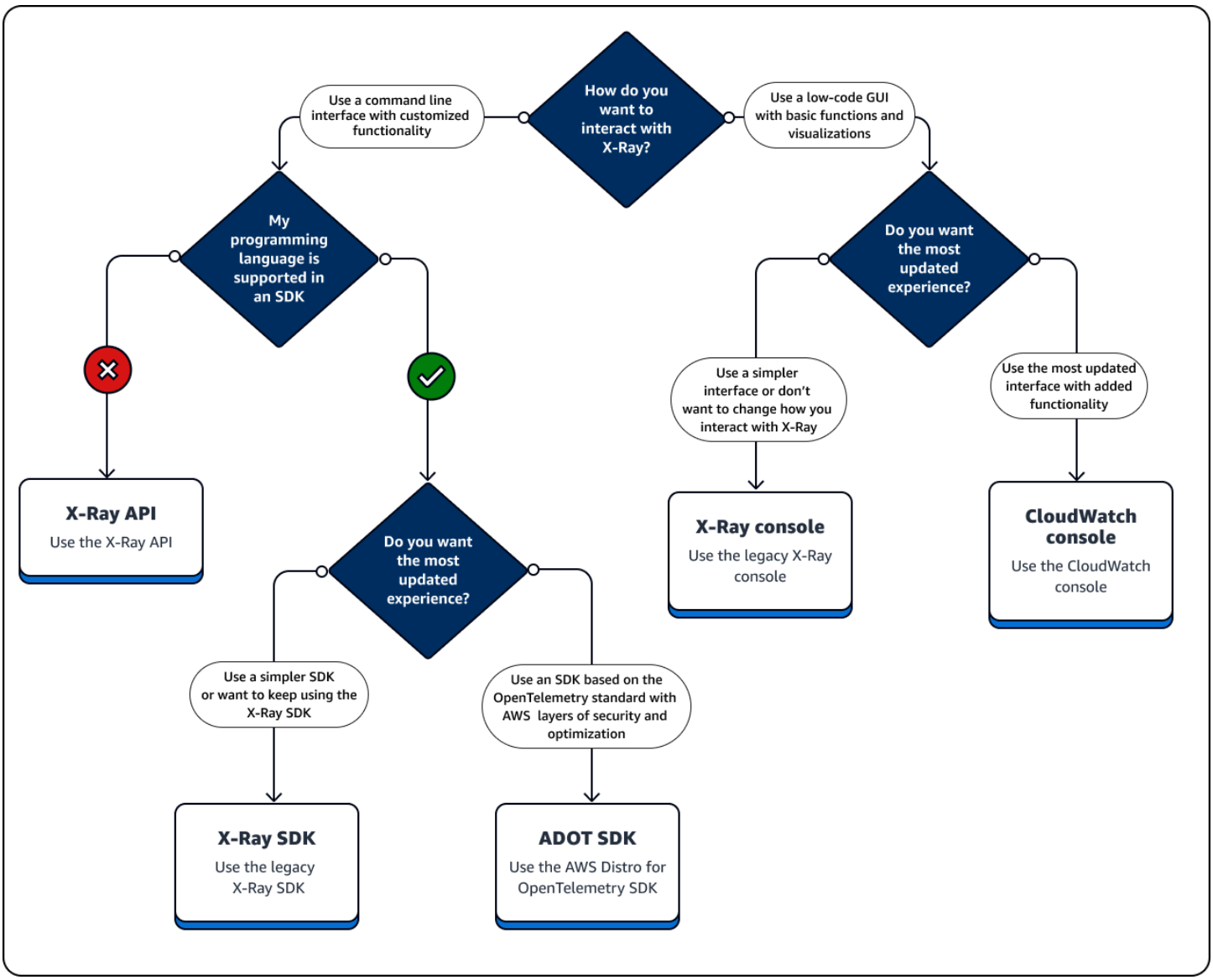
コンソールで使用できるデータとマップの例については、トレースデータを生成するために既に実装されているサンプルアプリケーションを起動します。数分で、トラフィックの生成、セグメントの X-Ray への送信、トレースとサービスマップの表示を行うことができます。

インターフェイスを選択する

AWS X-Ray は、アプリケーションの仕組みや、他の のサービスやリソースとどの程度うまくやり取りしているかに関するインサイトを提供できます。アプリケーションを計測または設定すると、X-Ray はアプリケーションがリクエストを処理するときにトレースデータを収集します。このトレースデータを分析して、パフォーマンスの問題の特定、エラーのトラブルシューティング、リソースの最適化を行うことができます。このガイドでは、以下のガイドラインに従って X-Ray を操作する方法について説明します。

- すぐに使用を開始する場合や、構築済みの視覚化を使用して基本的なタスクを実行できる AWS Management Console 場合は、 を使用します。
  - Amazon CloudWatch コンソールを選択すると、X-Ray コンソールのすべての機能を含む最新のユーザーエクスペリエンスが表示されます。
  - よりシンプルなインターフェイスが必要な場合や、X-Ray の操作方法を変更しない場合は、X-Ray コンソールを使用します。
- が提供するよりも多くのカスタムトレース、モニタリング、またはログ記録機能が必要な場合は、SDK AWS Management Console を使用します。
  - セキュリティ AWS と最適化のレイヤーを追加したオープンソース ADOT SDK に基づいてベンダーに依存しない SDK が必要な場合は、OpenTelemetrySDK を選択します。
  - よりシンプルな SDK が必要な場合、またはアプリケーションコードを更新しない場合は、X-Ray SDK を選択します。
- SDK がアプリケーションのプログラミング言語をサポートしていない場合は、X-Ray API オペレーションを使用します。

次の図は、X-Ray を操作する方法を選択するのに役立ちます。



インターフェイスタイプを調べる

- を使用する AWS Management Console
- SDK を使用する
- X-Ray API を使用する

を使用する AWS Management Console

最小限のコーディングでグラフィカルユーザーインターフェイス (GUI) AWS Management Console が必要な場合は、 を使用します。X-Ray を初めて使用するユーザーは、事前に構築された視覚化を

使用して、基本的なタスクを実行することで、すぐに開始できます。コンソールから直接以下を実行できます。

- X-Ray を有効にします。
- アプリケーションのパフォーマンスの概要を表示します。
- アプリケーションのヘルスステータスを確認します。
- 高レベルのエラーを特定します。
- 基本的なトレースの概要を表示します。

https://console.aws.amazon.com/cloudwatch/ の Amazon CloudWatch コンソールまたは https://console.aws.amazon.com/xray/home の X-Ray コンソールを使用して、X-Ray とやり取りできます。

Amazon CloudWatch コンソールを使用する

CloudWatch コンソールには、使いやすくするために X-Ray コンソールから再設計された新しい X-Ray 機能が含まれています。 CloudWatch コンソールを使用する場合は、X-Ray トレースデータとともに CloudWatch ログとメトリクスを表示できます。 CloudWatch コンソールを使用して、次のようなデータを表示および分析します。

- X-Ray トレース – アプリケーションがリクエストを処理する際に、アプリケーションに関連付けられたトレースを表示、分析、フィルタリングします。これらのトレースを使用して、高いレイテンシーを検出し、エラーをデバッグし、アプリケーションワークフローを最適化します。トレースマップとサービスマップを表示して、アプリケーションワークフローを視覚的に表示します。
- ログ – アプリケーションが生成するログを表示、分析、フィルタリングします。ログを使用してエラーをトラブルシューティングし、特定のログ値に基づいてモニタリングを設定します。
- メトリクス – リソースが出力するメトリクスを使用してアプリケーションのパフォーマンスを測定およびモニタリングするか、独自のメトリクスを作成します。これらのメトリクスをグラフとグラフで表示します。
- ネットワークとインフラストラクチャのモニタリング — コンテナ化されたアプリケーション、その他の AWS サービス、クライアントなど、インフラストラクチャの停止やヘルスとパフォーマンスについて、主要なネットワークをモニタリングします。
- 次の X-Ray コンソールを使用するセクションにリストされている X-Ray コンソールのすべての機能。

CloudWatch コンソールの詳細については、「Amazon の開始方法 CloudWatch」を参照してください。

https://console.aws.amazon.com/cloudwatch/ で Amazon CloudWatch コンソールにログインします。

X-Ray コンソールを使用する

X-Ray コンソールには、アプリケーションリクエストの分散トレースが用意されています。コンソールエクスペリエンスをシンプルにしたい場合や、アプリケーションコードを更新しない場合は、X-Ray コンソールを使用します。 AWS は X-Ray コンソールを開発しなくなりました。X-Ray コンソールには、インストルメント化されたアプリケーションの以下の機能が含まれています。

- Insights – アプリケーションのパフォーマンスの異常を自動的に検出し、根本的な原因を見つけます。Insights は、Insights の下の CloudWatch コンソールに含まれています。詳細については、「 で X-Ray Insights を使用する」を参照してくださいX-Ray コンソールの詳細。
- サービスマップ — アプリケーションとそのクライアント、リソース、サービス、依存関係との接続をグラフィカルに表示します。
- トレース — アプリケーションがリクエストを処理するときに生成されるトレースの概要を表示します。トレースデータを使用して、HTTPレスポンスやレスポンスタイムなどの基本的なメトリクスに対するアプリケーションの動作を理解します。
- 分析 — 応答時間分布のグラフを使用して、トレースデータを解釈、調査、分析します。
- 設定 — カスタマイズしたトレースを作成して、以下のデフォルト設定を変更します。
  - サンプリング — トレース情報用にアプリケーションをサンプリングする頻度を定義するルールを作成します。詳細については、「 でサンプリングルールを設定するX-Ray コンソールの詳細」を参照してください。
  - 暗号化 – を使用して監査または無効化できるキーを使用して、保管中のデータを暗号化します AWS Key Management Service。
  - グループ – フィルター式を使用して、URL の名前や応答時間などの一般的な特徴を持つトレースのグループを定義します。詳細については、「グループの設定」を参照してください。

https://console.aws.amazon.com/xray/home で X-Ray コンソールにログインします。

X-Ray コンソールの詳細

X-Ray コンソールを使用して、アプリケーションが処理するリクエストのサービスおよび関連するトレースのマップを表示し、トレースを X-Ray に送信する方法に影響するグループとサンプリングルールを設定します。

> **Note**
>
> X-Ray Service マップと CloudWatch ServiceLens マップは、Amazon CloudWatch コンソール内の X-Ray トレースマップにまとめられています。CloudWatch コンソールを開き、左側のナビゲーションペインから X-Ray トレースの下にあるトレースマップを選択します。CloudWatch に Application Signals が含まれるようになりました。これにより、アプリケーションサービス、クライアント、Synthetics Canary、およびサービスの依存関係を検出してモニタリングできます。Application Signals を使用すると、サービスのリストやビジュアルマップを確認したり、サービスレベル目標 (SLO) に基づくヘルスメトリクスを表示したり、ドリルダウンして相関関係のある X-Ray トレースを確認したりして、より詳細なトラブルシューティングを行うことができます。

プライマリ X-Ray コンソールページはトレースマップです。トレースマップは、アプリケーションによって生成されたトレースデータから X-Ray が生成する JSON サービスグラフを視覚的に表現したものです。マップは、リクエストを処理するアカウント内の各アプリケーションのサービスノード、リクエストの送信元を示すアップストリームクライアントノード、リクエストの処理中にアプリケーションが使用するウェブサービスとリソースを示すダウンストリームサービスノードで構成されます。他にも、トレースとトレースの詳細を表示したり、グループやサンプリングルールを設定したりするためのページがあります。

X-Ray のコンソールエクスペリエンスを表示し、以下のセクションで CloudWatch コンソールと比較します。

X-Ray トレースマップを使用する

X-Ray トレースマップを表示して、エラーが発生しているサービス、レイテンシーの高い接続、または失敗したリクエストのトレースを識別します。

> **Note**
>
> CloudWatch に Application Signals が含まれるようになりました。これにより、アプリケーションサービス、クライアント、Synthetics Canary、およびサービスの依存関係を検出して

モニタリングできます。Application Signals を使用すると、サービスのリストやビジュアルマップを確認したり、サービスレベル目標 (SLO) に基づくヘルスメトリクスを表示したり、ドリルダウンして相関関係のある X-Ray トレースを確認したりして、より詳細なトラブルシューティングを行うことができます。
X-Ray サービスマップと CloudWatch ServiceLens マップは、Amazon CloudWatch コンソール内の X-Ray トレースマップに結合されます。CloudWatch コンソールを開き、左側のナビゲーションペインから X-Ray トレースの下にあるトレースマップを選択します。

トレースマップの表示

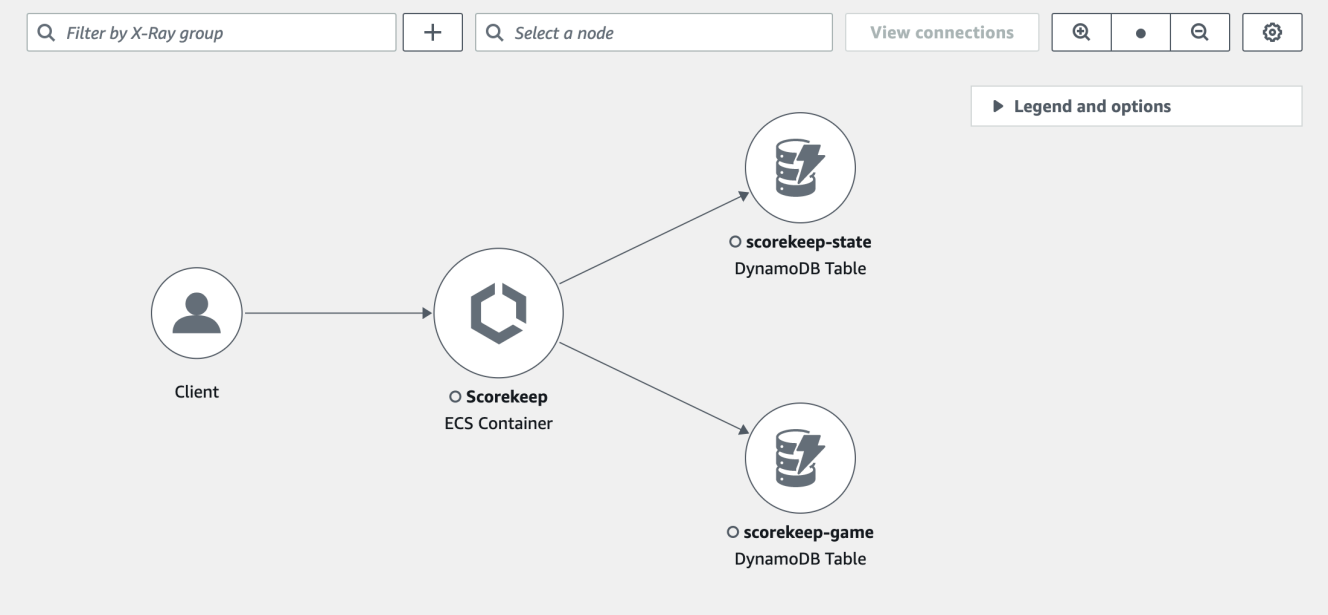
トレースマップは、アプリケーションによって生成されたトレースデータを視覚的に表現したものです。リクエストを処理するサービスノード、リクエストの送信元を示すアップストリームクライアントノード、リクエストの処理中にアプリケーションが使用するウェブサービスとリソースを示すダウンストリームサービスノードがマップに表示されます。

トレースマップには、Amazon SQS と Lambda を使用するイベント駆動型アプリケーション全体のトレースの接続されたビューが表示されます。詳細については、次のトレースイベント駆動型アプリケーションセクションを参照してください。トレースマップは、クロスアカウントトレース もサポートし、複数のアカウントのノードを 1 つのマップに表示します。

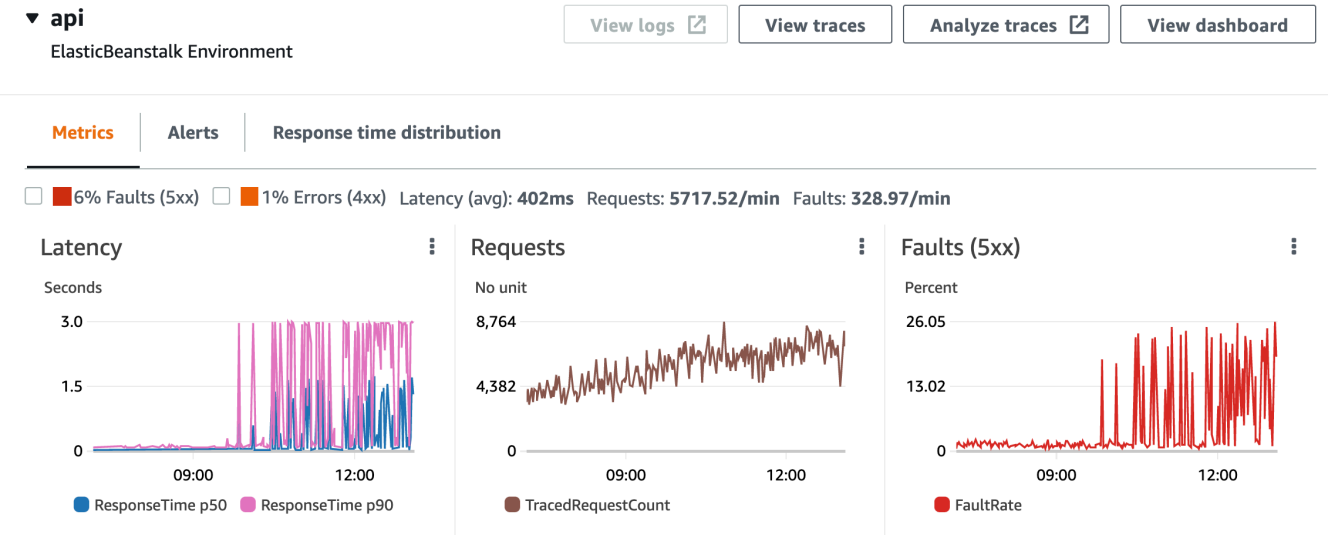
CloudWatch console

CloudWatch コンソールでトレースマップを表示するには

1. CloudWatch コンソールを開きます。左側のナビゲーションペインの X-Ray トレースセクションでトレースマップを選択します。



2. そのノードのリクエスト、または 2 つのノード間のエッジを表示するサービスノードを選択して、やり取りされた接続のリクエストを表示します。
3. メトリクス、アラート、応答時間分布のタブなど、追加情報がトレースマップの下に表示されます。メトリクスタブで、各グラフ内の範囲を選択して詳細を表示するか、障害またはエラーオプションを選択してトレースをフィルタリングします。[応答時間の分布] タブでは、応答時間でトレースをフィルタリングするグラフ内の範囲を選択します。



4. [トレースを表示] を選択してトレースを表示するか、フィルターが適用されている場合は [フィルタリングされたトレースの表示] を選択します。
5. ログを表示 を選択して、選択したノードに関連付けられた CloudWatch ログを表示します。すべてのトレースマップノードがログの表示をサポートしているわけではありません。詳細については、「トラブルシューティング CloudWatch ログ」を参照してください。

トレースマップは、各ノード内の問題を色で囲んで示します。

- 赤は、サーバー障害 (500 系のエラー)
- 黄は、クライアントエラー (400 系のエラー)
- 紫は、スロットリングエラー (429 リクエストが多すぎる)

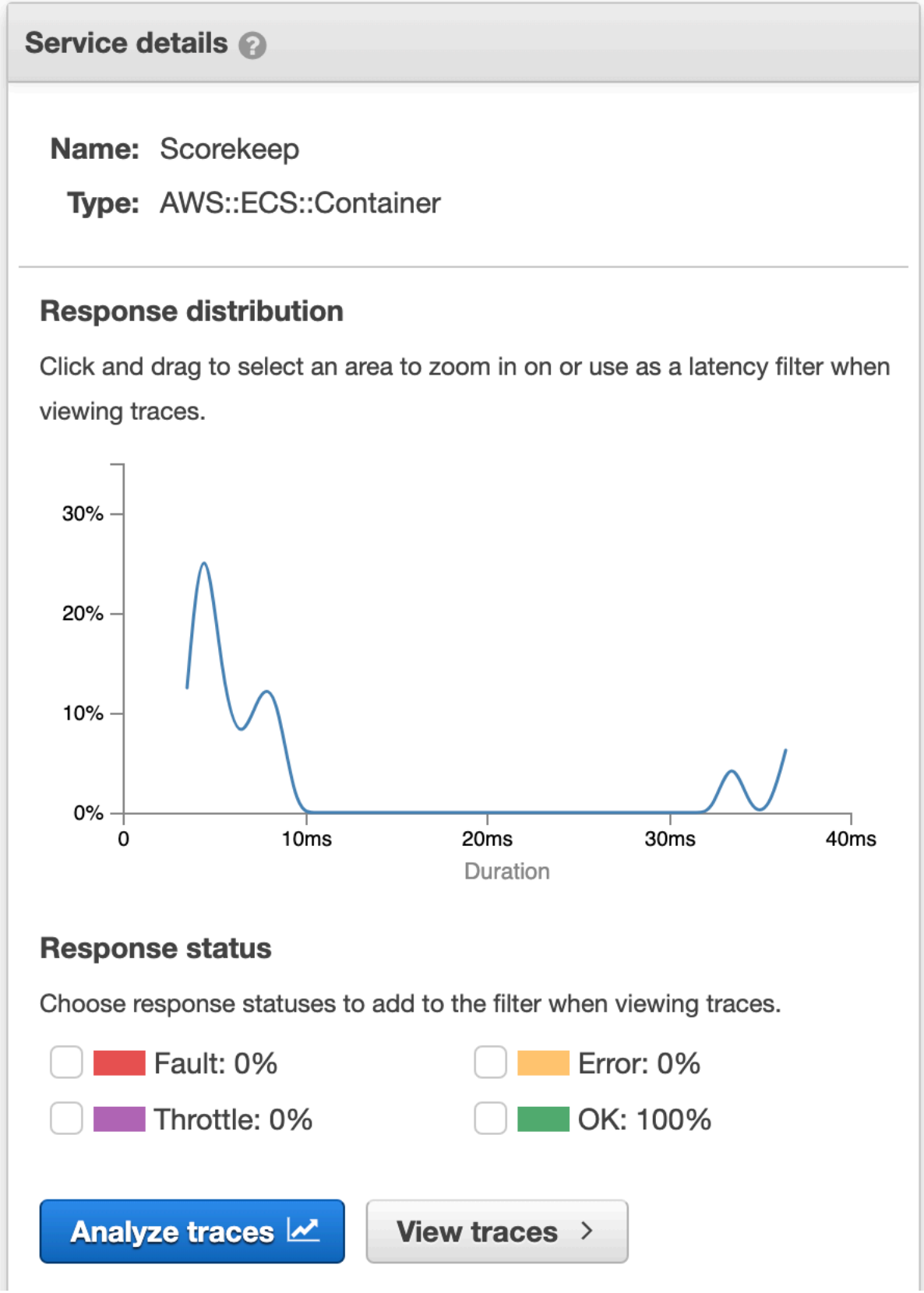
トレースマップが大きい場合は、画面上のコントロールまたはマウスを使用して、マップを拡大/縮小して移動します。

X-Ray console

サービスマップを表示するには

1. [X-Ray console (X-Ray コンソール)] を開きます。デフォルトでは、サービスマップが表示されます。左側のナビゲーションペインからサービスマップを選択することもできます。

2. そのノードのリクエスト、または 2 つのノード間のエッジを表示するサービスノードを選択して、やり取りされた接続のリクエストを表示します。
3. レスポンス分散ヒストグラムを使用して、トレースを期間でフィルタリングし、トレースを表示するステータスコードを選択します。[View traces (トレースの表示)] を選択し、フィルタ式を適用してトレースリストを開きます。ディストリビューションヒストグラムの詳細については、「」を参照してください???。


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

このサービスマップは、各ノードの状態をエラーと障害に対する正常な呼び出しの比率に基づいて色分けしたものです。

- 緑は、正常な呼び出し
- 赤は、サーバー障害 (500 系のエラー)
- 黄は、クライアントエラー (400 系のエラー)
- 紫は、スロットリングエラー (429 リクエストが多すぎる)

サービスマップが大きい場合は、画面のコントロールまたはマウスを使用して、マップを拡大/縮小したり移動したりします。

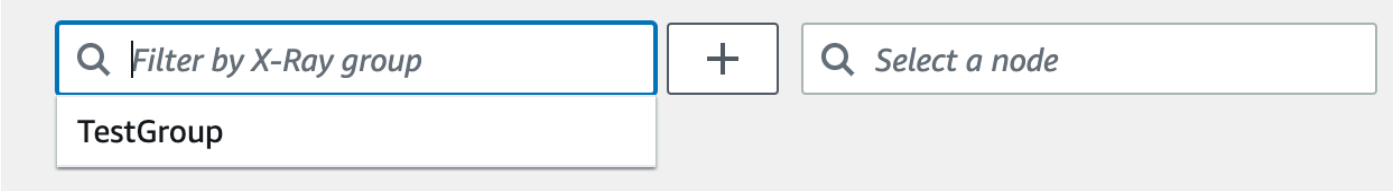
> **Note**
>
> X-Ray トレースマップには、最大 10,000 個のノードを表示できます。まれに、サービスノードの総数がこの制限を超えると、エラーが表示され、コンソールに完全なトレースマップを表示できないことがあります。

グループによるトレースマップのフィルタリング

フィルター式を使用すると、グループに含めるトレースの基準を定義できます。フィルター式の詳細については、「フィルター式の使用」を参照してください。次に、次のステップを使用して、トレースマップにその特定のグループを表示します。

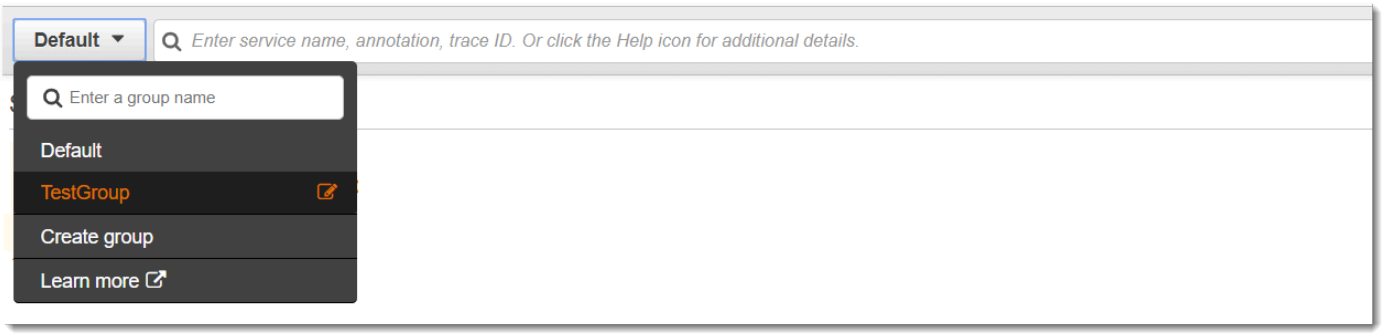
CloudWatch console

トレースマップの左上にあるグループフィルターからグループ名を選択します。



X-Ray console

検索バーの左側にあるドロップダウンメニューからグループ名を選択します。



これで、サービスマップがフィルタリングされ、選択したグループのフィルター式と一致するトレースが表示されます。

トレースマップの凡例とオプション

トレースマップには、凡例とマップ表示をカスタマイズするためのいくつかのオプションが含まれています。

CloudWatch console

マップの右上にある [凡例とオプション] ドロップダウンを選択します。以下のノード内に表示する内容を選択します。

- メトリクスには、選択した時間範囲の平均応答時間と 1 分あたりに送信されたトレース数が表示されます。
- ノードには、各ノード内のサービスアイコンが表示されます。

[設定] ペインから追加のマップ設定を選択します。このペインには、マップの右上にある歯車アイコンからアクセスできます。これらの設定には、各ノードのサイズを決定するために使用するメトリクスや、マップに表示する Canary の選択などがあります。

X-Ray console

サービスマップの凡例を表示するには、マップの右上にある [マップの凡例] リンクを選択します。サービスマップオプションは、トレースマップの右下で選択できます。これには、以下が含まれます。

- サービスアイコン 各ノード内の表示内容を切り替えて、サービスアイコンを表示するか、または選択した時間範囲の平均応答時間と 1 分あたりに送信されたトレース数を表示します。
- ノードサイズ : なし すべてのノードを同じサイズに設定します。

- ノードサイズ : ヘルス エラー、障害、スロットリングされたリクエストなど、影響を受けるリクエストの数に応じてノードのサイズを設定します。
- ノードサイズ : トラフィック リクエストの合計数に応じてノードのサイズを設定します。

トレースとトレースの詳細を表示する

X-Ray コンソールの [トレース] ページを使用して、URL、レスポンスコード、トレース概要のその他のデータでトレースを検索します。トレースリストからトレースを選択すると、トレースの詳細ページに、選択したトレースに関連付けられているサービスノードのマップとトレースセグメントのタイムラインが表示されます。

トレースの表示

CloudWatch console

CloudWatch コンソールでトレースを表示するには

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開きます。
2. 左側のナビゲーションペインで、X-Ray トレース を選択し、トレース を選択します。グループでフィルタリングしたり、フィルター式を入力したりできます。フィルター式は、ページの下部にあるトレースセクションに表示されるトレースをフィルタリングします。詳細については、「フィルター式を使用する」を参照してください。

   または、サービスマップを使用して特定のサービスノードに移動し、トレースを表示することもできます。これにより、クエリが既に適用されているトレースページが開きます。
3. [クエリリファイナー] セクションでクエリを絞り込みます。共通の属性でトレースをフィルタリングするには、 でクエリを絞り込むの横にある下矢印からオプションを選択します。オプションは以下のとおりです。

   - Node – サービスノードでトレースをフィルタリングします。
   - リソース ARN – トレースに関連付けられたリソースでトレースをフィルタリングします。これらのリソースの例としては、Amazon Elastic Compute Cloud (Amazon EC2) インスタンス、 AWS Lambda 関数、 Amazon DynamoDB テーブルなどがあります。
   - ユーザー – ユーザー ID でトレースをフィルタリングします。
   - エラーの根本原因メッセージ — エラーの根本原因でトレースをフィルタリングします。
   - URL – アプリケーションで使用される URL パスでトレースをフィルタリングします。

- HTTP ステータスコード – アプリケーションから返された HTTP ステータスコードでトレースをフィルタリングします。カスタムレスポンスコードを指定するか、以下から選択できます。
  - `200` – リクエストは成功しました。
  - `401` – リクエストに有効な認証情報がありません。
  - `403` – リクエストに有効なアクセス許可がありませんでした。
  - `404` – サーバーはリクエストされたリソースを見つけることができませんでした。
  - `500` – サーバーで予期しない状態が発生し、内部エラーが発生しました。

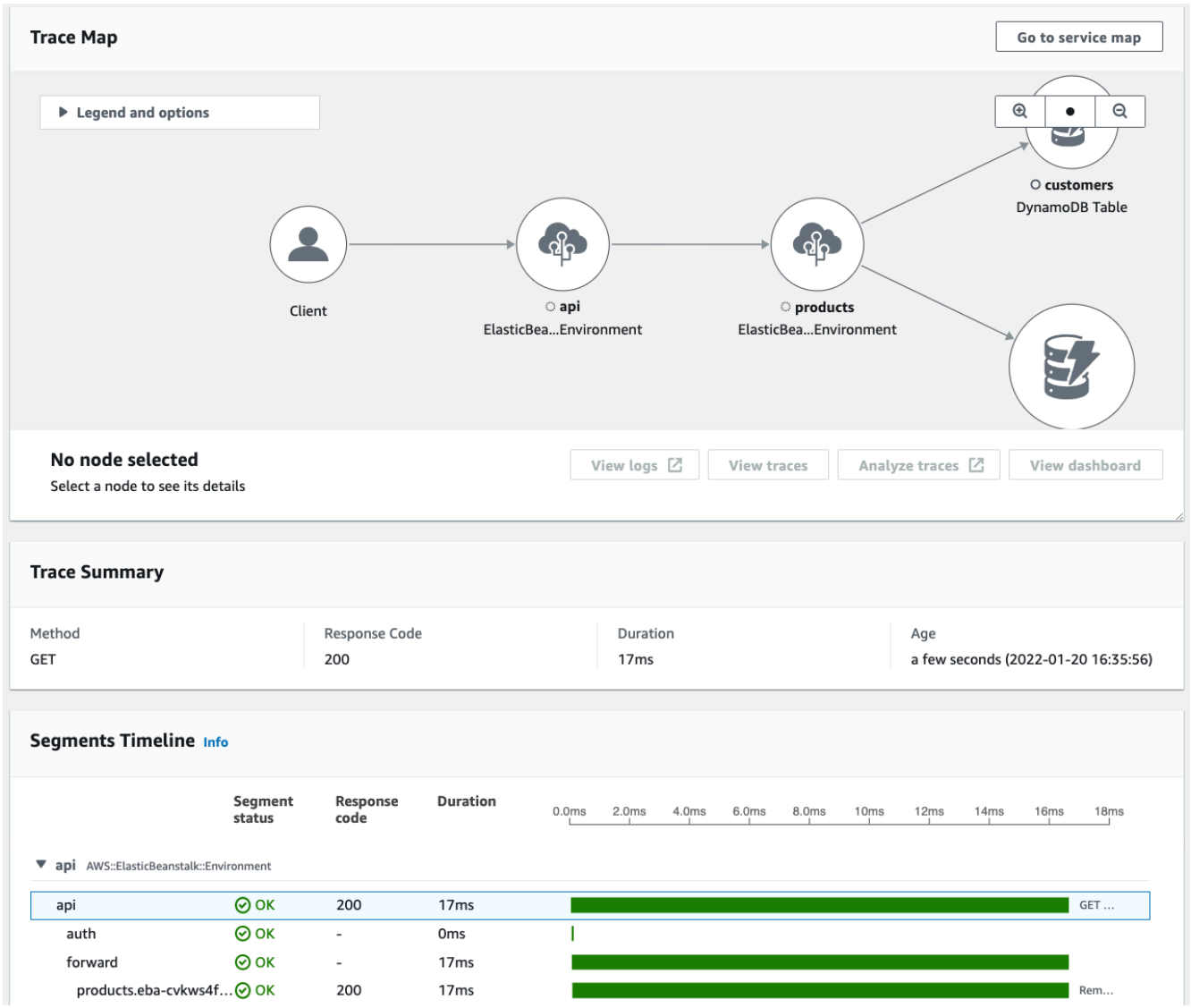
1 つ以上のエントリを選択し、クエリに追加 を選択して、ページ上部のフィルター式に追加します。

4. 単一のトレースを検索するには、トレース ID をクエリフィールドに直接入力します。X-Ray 形式または World Wide Web Consortium (W3C) 形式を使用できます。例えば、 AWS Distro for OpenTelemetryを使用して作成されたトレースは W3C 形式です。

   > **Note**
   >
   > W3C-formatトレース ID で作成されたトレースをクエリすると、コンソールには一致するトレースが X-Ray 形式で表示されます。例えば、W3C 形式で `4efaaf4d1e8720b39541901950019ee5`をクエリすると、コンソールに X-Ray に相当する が表示されます`1-4efaaf4d-1e8720b39541901950019ee5`。

5. [クエリを実行] を選択すると、いつでもページ下部の [トレース] セクションに一致するトレースのリストが表示されます。
6. 1 つのトレースのトレース詳細ページを表示するには、リストからトレース ID を選択します。

   次の図は、トレースに関連付けられたサービスノードと、トレースを構成するセグメントによって取得されるパスを表すノード間のエッジを含むトレースマップを示しています。トレースの概要は、トレースマップ の後に続きます。概要には、サンプルGETオペレーション、レスポンスコード 、トレースの実行にかかった時間、およびリクエストの経過時間に関する情報が含まれます。セグメントタイムラインは、トレースセグメントとサブセグメントの期間を示すトレースサマリーに従います。



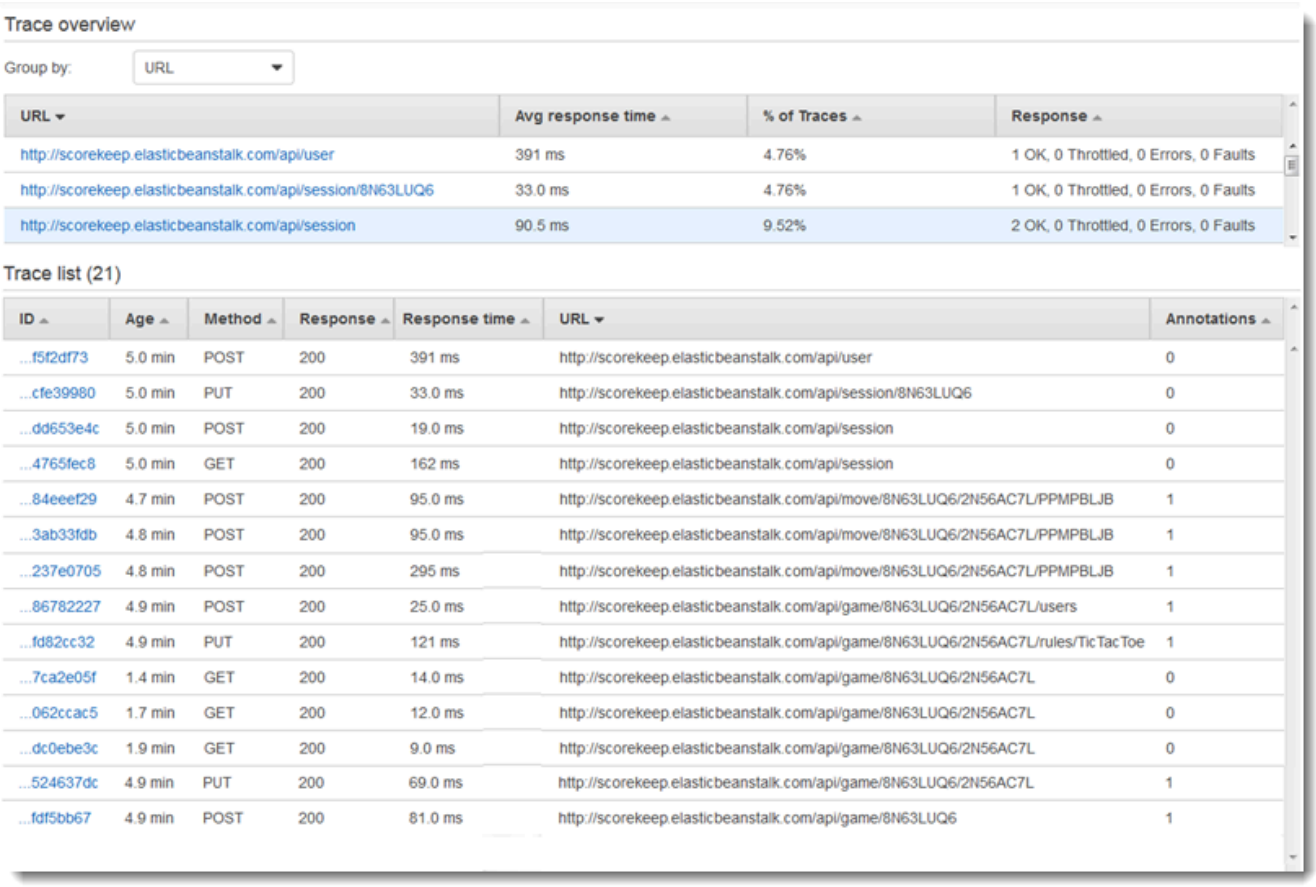
Amazon SQS と Lambda を使用するイベント駆動型アプリケーションがある場合、トレースマップ で各リクエストのトレースの接続されたビューを確認できます。マップでは、メッセージプロデューサーからのトレースは AWS Lambda コンシューマーからのトレースにリンクされ、破線のエッジとして表示されます。イベント駆動型アプリケーションの詳細については、「」を参照してください[イベント駆動型アプリケーションをトレースする]。

トレースページとトレースの詳細ページでは、クロスアカウントトレースもサポートされています。クロスアカウントトレースでは、トレースリストと 1 つのトレースマップ内の複数のアカウントのトレースを一覧表示できます。詳細については、「[クロスアカウントトレース]」を参照してください。

X-Ray console

X-Ray コンソールでトレースを表示するには

1. X-Ray コンソールのトレースページを開きます。トレースの概要パネルには、エラーの根本原因 、ResourceARN 、 などの一般的な機能別にグループ化されたトレースのリストが表示されますInstanceId。 ResourceARN
2. トレースのグループ化されたセットを表示する共通機能を選択するには、グループ化 の横にある下矢印を展開します。次の図は、 の URL 別にグループ化されたトレースのトレースの概要とAWS X-Ray サンプルアプリケーション、関連するトレースのリストを示しています。



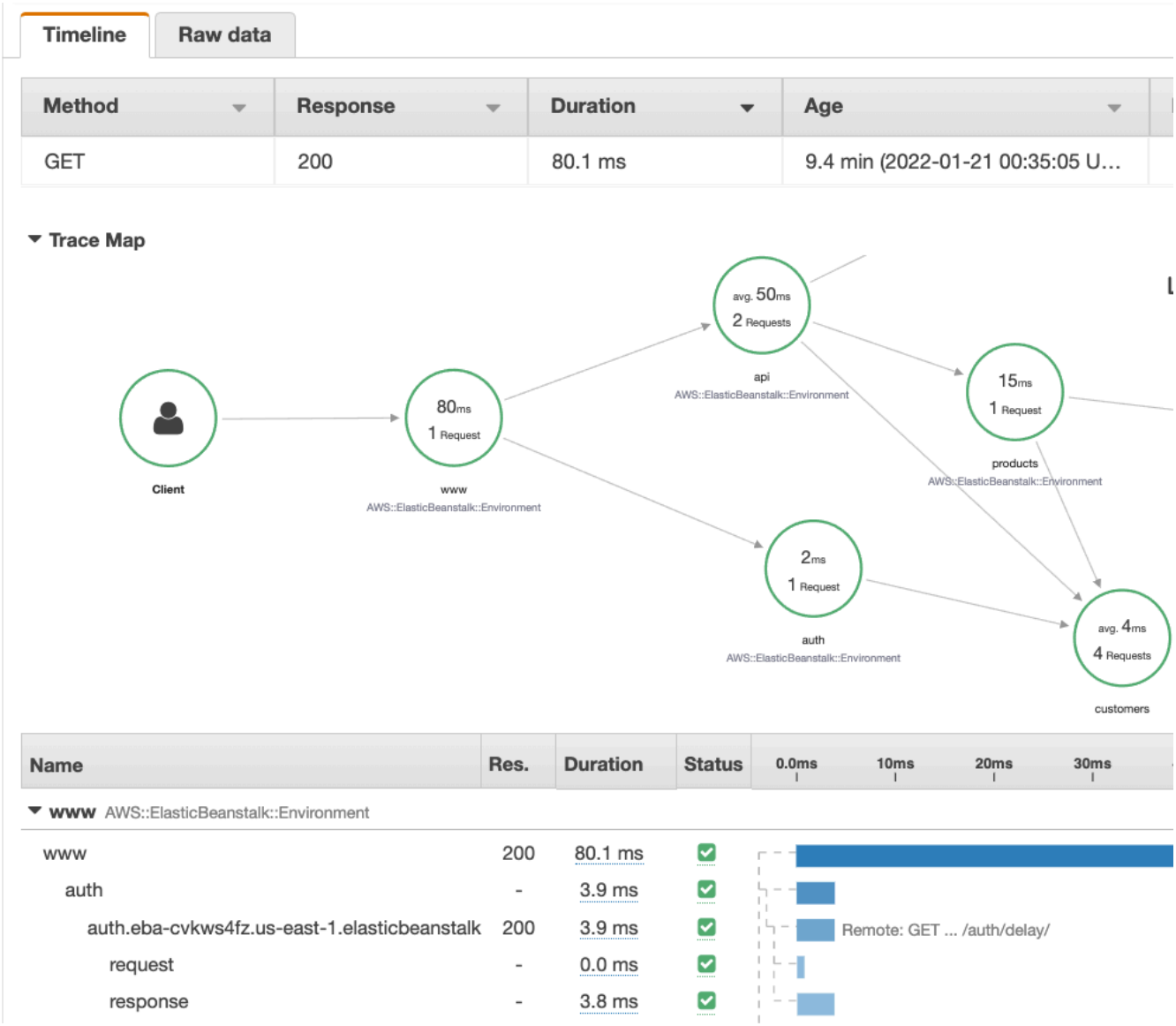
3. トレースの ID を選択すると、トレースリスト の下に表示されます。ナビゲーションペインでサービスマップを選択して、特定のサービスノードのトレースを表示することもできます。その後、そのノードに関連付けられているトレースを表示できます。

   Timeline タブにはトレースのリクエストフローが表示され、以下が含まれます。

   - トレース内の各セグメントのパスのマップ。

- セグメントがトレースマップ内のノードに到達するまでにかかる時間。
- トレースマップ内のノードに対して行われたリクエストの数。

次の図は、サンプルアプリケーションに対して行われたGETリクエストに関連付けられたトレースマップの例を示しています。矢印は、各セグメントがリクエストを完了するために取ったパスを示しています。サービスノードには、リクエスト中に行われたGETリクエストの数が表示されます。



タイムラインタブの詳細については、次の「トレースタイムラインの探索」セクションを参照してください。

Raw データタブには、トレースに関する情報と、トレースを構成するセグメントとサブセグメントが JSON 形式で表示されます。この情報には以下が含まれます。

- タイムスタンプ
- 一意の ID
- セグメントまたはサブセグメントに関連付けられたリソース
- セグメントまたはサブセグメントのソースまたはオリジン
- HTTP リクエストからのレスポンスなど、アプリケーションへのリクエストに関する追加情報

トレースのタイムラインの探索

タイムラインセクションには、タスクの完了に使用された時間に対応する水平バーの横にあるセグメントとサブセグメントの階層が表示されます。リスト内の最初のエントリは、1 回のリクエストに対してサービスによって記録されたすべてのデータを表すセグメントです。サブセグメントはインデントされ、セグメントの後に一覧表示されます。列には、各セグメントに関する情報が含まれます。

CloudWatch console

CloudWatch コンソールでは、セグメントタイムラインに次の情報が表示されます。

- 最初の列: 選択したトレースのセグメントとサブセグメントを一覧表示します。
- セグメントステータス列: 各セグメントとサブセグメントのステータス結果を一覧表示します。
- レスポンスコード列: HTTP レスポンスステータスコードを、セグメントまたはサブセグメントによって行われたブラウザリクエストに一覧表示します。
- Duration 列: セグメントまたはサブセグメントが実行された時間を一覧表示します。
- Hosted in 列: 必要に応じて、セグメントまたはサブセグメントが実行されている名前空間または環境を一覧表示します。詳細については、「https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/AppSignals-StandardMetrics.html#AppSignals-StandardMetrics-Dimensions」を参照してください。
- 最後の列: タイムライン内の他のセグメントまたはサブセグメントに関連して、セグメントまたはサブセグメントが実行された期間に対応する水平バーを表示します。

セグメントとサブセグメントのリストをサービスノード別にグループ化するには、ノード別にグループ化 をオンにします。

X-Ray console

トレースの詳細ページでタイムラインタブを選択すると、トレースを構成する各セグメントとサブセグメントのタイムラインが表示されます。

X-Ray コンソールでは、タイムラインは次の情報を提供します。

- 名前列: トレース内のセグメントとサブセグメントの名前を一覧表示します。
- Res. 列: HTTP レスポンスステータスコードを、セグメントまたはサブセグメントによって行われたブラウザリクエストに一覧表示します。
- Duration 列: セグメントまたはサブセグメントが実行された時間を一覧表示します。
- ステータス列: セグメントまたはサブセグメントのステータスの結果を一覧表示します。
- 最後の列: タイムライン内の他のセグメントまたはサブセグメントに関連して、セグメントまたはサブセグメントが実行された期間に対応する水平バーを表示します。

コンソールがタイムラインの生成に使用する未加工のトレースデータを表示するには、未加工のデータタブを選択します。raw データには、トレースに関する情報と、トレースを構成するセグメントとサブセグメントが JSON 形式で表示されます。この情報には以下が含まれます。

- タイムスタンプ
- 一意の ID
- セグメントまたはサブセグメントに関連付けられたリソース
- セグメントまたはサブセグメントのソースまたはオリジン
- HTTP リクエストからのレスポンスなど、アプリケーションへのリクエストに関する追加情報。

計測された AWS SDK、、HTTPまたはクライアントを使用して外部リソースを呼びSQL出すと、X-Ray SDK はサブセグメントを自動的に記録します。X-Ray SDK を使用して、任意の関数またはコードブロックのカスタムサブセグメントを記録することもできます。カスタムサブセグメントが開いている間に記録される追加のサブセグメントは、カスタムサブセグメントの子になります。

セグメントの詳細を表示する

トレースタイムライン からセグメントの名前を選択して、その詳細を表示します。

セグメントの詳細パネルには、概要 、リソース 、注釈 、メタデータ 、例外 、および SQL タブが表示されます。 以下が適用されます。

- [概要] タブには、リクエストと応答に関する情報が表示されます。情報には、名前、開始時刻、終了時刻、期間、リクエスト URL、リクエストオペレーション、リクエストレスポンスコード、エラーと障害が含まれます。
- セグメントのリソースタブには、X-Ray SDK からの情報と、アプリケーションを実行している AWS リソースに関する情報が表示されます。X-Ray SDK の Amazon EC2 AWS Elastic Beanstalk、または Amazon ECS プラグインを使用して、サービス固有のリソース情報を記録します。プラグインの詳細については、「」の「サービスプラグイン」セクションを参照してくださいいX-Ray SDK for Java の設定。
- 残りのタブには、セグメントに記録された注釈 、メタデータ 、および例外が表示されます。例外は、計測されたリクエストから生成されると自動的にキャプチャされます。注釈とメタデータには、X-Ray SDK が提供するオペレーションを使用して記録する追加情報が含まれています。セグメントに注釈またはメタデータを追加するには、X-Ray SDK を使用します。詳細については、「」の AWS X-Ray SDKs」に記載されている言語固有のリンクを参照してくださいのアプリケーションを計測する AWS X-Ray。

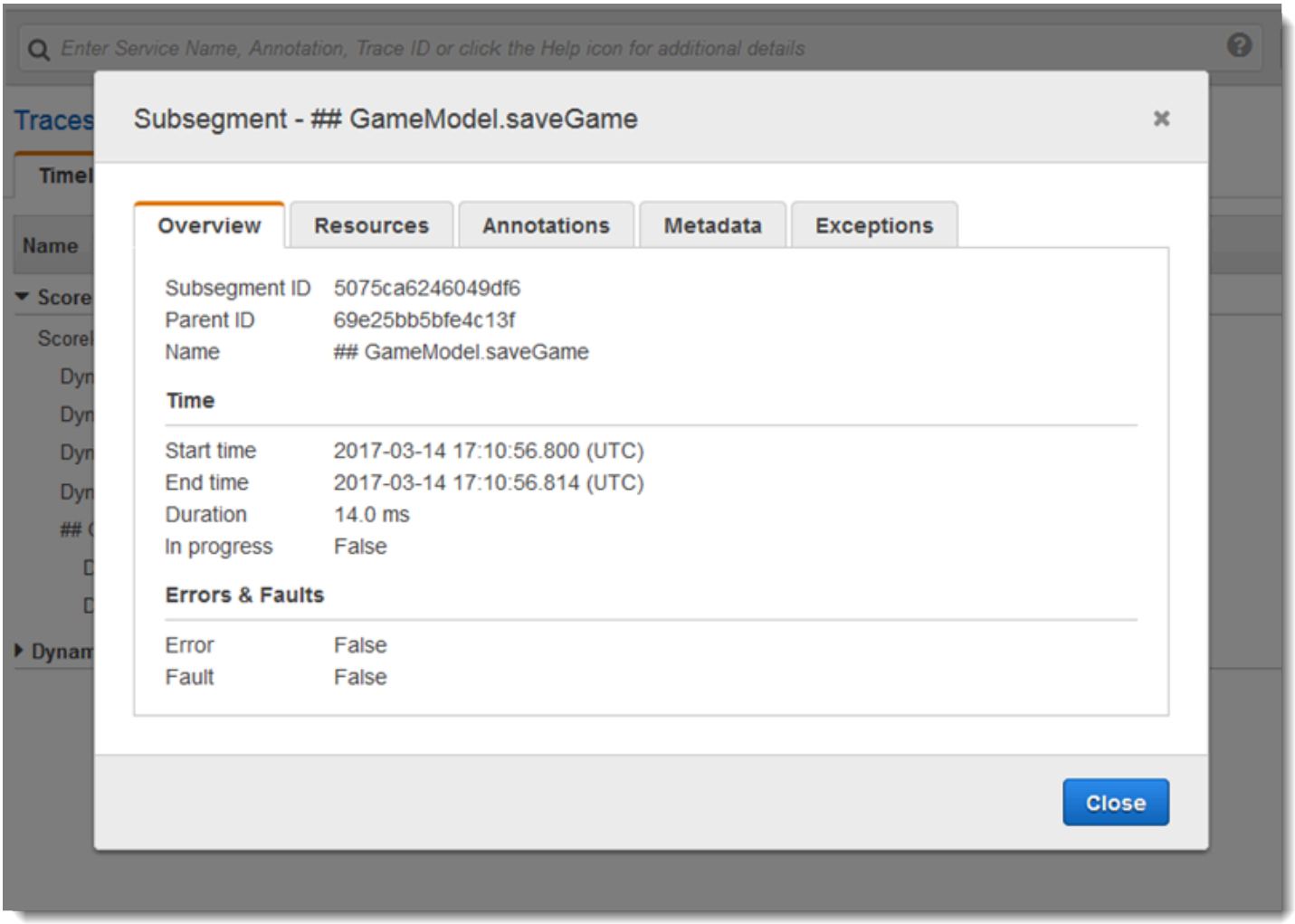
サブセグメントの詳細を表示する

トレースタイムラインから、詳細を表示するサブセグメントの名前を選択します。

- 概要タブには、リクエストとレスポンスに関する情報が含まれています。これには、名前、開始時刻、終了時刻、期間、リクエスト URL、リクエストオペレーション、リクエストレスポンスコード、エラーと障害が含まれます。計測されたクライアントを使用して生成されたサブセグメントについては、[概要] タブにアプリケーションの視点からのリクエストと応答に関する情報が含まれています。
- サブセグメントのリソースタブには、 AWS サブセグメントの実行に使用されたリソースの詳細が表示されます。例えば、リソースタブには、 AWS Lambda 関数 ARN、DynamoDB テーブルに関する情報、呼び出されるオペレーション、およびリクエスト ID が含まれる場合があります。
- 残りのタブには、サブセグメントに記録された注釈 、メタデータ 、および例外が表示されます。例外は、計測されたリクエストから生成されると自動的にキャプチャされます。注釈とメタデータには、X-Ray SDK が提供するオペレーションを使用して記録する追加情報が含まれています。X-Ray SDK を使用して、セグメントに注釈またはメタデータを追加します。詳細については、「」の「 SDK を使用した AWS X-Ray アプリケーションの計測 SDKs」に記載されている言語固有のリンクを参照してくださいのアプリケーションを計測する AWS X-Ray。

カスタムサブセグメントの場合、[概要] タブには記録するコードまたは関数の領域を指定するために設定できるサブセグメントの名前が表示されます。詳細については、「」の AWS X-Ray SDKs」に記載されている言語固有のリンクを参照してくださいX-Ray SDK for Java を使用したカスタムサブセグメントの生成。

次の図は、カスタムサブセグメントの概要タブを示しています。概要には、サブセグメント ID、親 ID、名前、開始時刻と終了時刻、期間、ステータス、エラーまたは障害が含まれます。



カスタムサブセグメントのメタデータタブには、そのサブセグメントで使用されるリソースに関する情報が JSON 形式で含まれます。

フィルター式を使用する

フィルター式を使用して、特定のリクエスト、サービス、2 つのサービス間の接続 (エッジ) 、または条件を満たすリクエストのトレースマップまたはトレースを表示します。X-Ray にはフィルタ式

言語があり、リクエストヘッダー、レスポンスステータス、元セグメントのインデックス付きフィールドのデータに基づいて、リクエスト、サービス、エッジをフィルタリングできます。

X-Ray コンソールで表示するトレースの期間を選択すると、コンソールが表示できる以上の結果が得られることがあります。右上隅には、スキャンしたトレースの数と使用可能なトレースが他にもあるかどうかがコンソールに表示されます。フィルター式を使用して、検索するトレースだけに結果を絞り込むことができます。

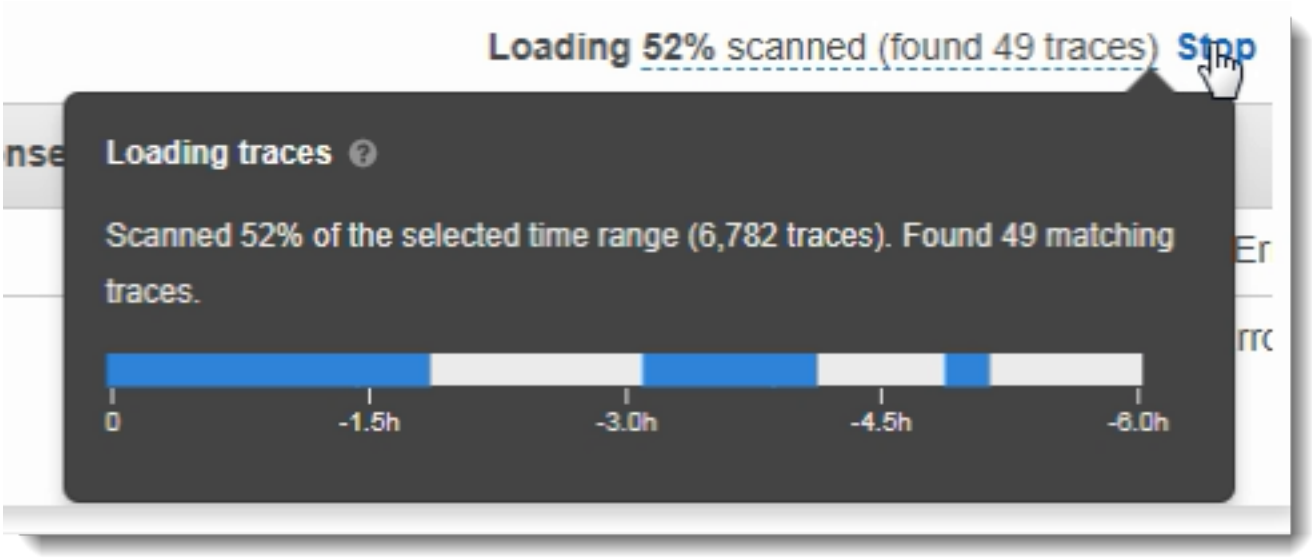
フィルタ式の詳細

トレースマップでノードを選択すると、コンソールはノードのサービス名と選択内容に基づいて存在するエラーのタイプに基づいてフィルター式を作成します。パフォーマンスの問題を示すトレースや特定のリクエストに関連するトレースを見つけるには、コンソールが提供する式を調整するか、独自の式を作成します。X-Ray SDK で注釈を追加する場合は、注釈キーまたはキーの値に基づいてフィルターを適用することもできます。

> **Note**
>
> トレースマップで相対時間範囲を選択し、ノードを選択すると、コンソールは時間範囲を絶対開始時刻と終了時刻に変換します。ノードのトレースが検索結果に確実に表示され、ノードがアクティブでないときのスキャン時間を避けるために、時間範囲にはノードがトレースを送信した時間だけが含まれます。現在の時刻を基準にして検索するには、トレースページで相対的な時間範囲に戻って再度スキャンすることができます。

コンソールが表示できるものより多くの結果がまだある場合、コンソールには一致したトレースの数とスキャンされたトレースの数が表示されます。表示される割合は、スキャンされた選択済みの時間枠の割合です。結果に表示されているすべての一致するトレースを確認するには、フィルタ式をさらに絞り込むか、より短い時間枠を選択します。

一番新しい結果を得るために、コンソールは時間範囲の終わりにスキャンを開始し、逆方向に動作します。多数のトレースがあるが結果が少ない場合、コンソールは時間範囲をチャンクに分割し、それらを並行してスキャンします。進行状況バーには、スキャンされた時間範囲の一部が表示されます。



グループでフィルター式を使用する

グループは、フィルタ式で定義されるトレースのコレクションです。グループを使用して、追加のサービスグラフを生成し、Amazon CloudWatch メトリクスを指定できます。

グループは名前または Amazon リソースネーム (ARN) で識別され、フィルタ式を含みます。サービスは着信トレースを式と比較し、それに応じてそれらを保管します。

フィルタ式検索バーの左側にあるドロップダウンメニューを使用して、グループを作成および変更できます。

> **Note**
>
> グループの認定中に、サービスでエラーが検出された場合、そのグループは着信トレースの処理に含まれなくなり、エラーメトリクスが記録されます。

グループの詳細については、グループを設定するを参照してください。

フィルタ式の構文

フィルタ式にキーワード、単項またはバイナリの演算子、値を追加して比較することができます。

```
keyword operator value
```

演算子が異なる場合は、異なるタイプのキーワードを使用できます。たとえば、`responsetime` は、数値型キーワードを指し、数値に関する演算子と比較することができます。

Example - 応答時間が 5 秒を超えたリクエスト

```
responsetime > 5
```

AND 演算子および OR 演算子を使用して、複合式内で複数の式を結合できます。

Example - 総所要時間が 5〜8 秒のリクエスト

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

キーワードおよび演算子をシンプルにすると、トレースレベルでのみ問題を見つけることができます。エラーによってダウンロードが発生したが、アプリケーションによって処理され、ユーザーに返らない場合、error の検索では見つけることができません。

ダウンストリームの問題があるトレースを見つけるには、複雑なキーワード service()と を使用できますedge()。これらのキーワードを使用して、すべてのダウンストリームノード、単一のダウンストリームノード、または 2 つのノード間のエッジにフィルタ式を適用することができます。これらのキーワードの詳細については、次の「複雑なキーワード」セクションを参照してください。さらに詳細にする場合は、「id() 関数」を使用して、タイプごとにサービスおよびエッジをフィルタリングすることができます。詳細については、次の ID 関数セクションを参照してください。

ブール型キーワード

ブール値のキーワード値は true または false です。これらのキーワードを使用して、エラーの原因となったトレースを見つけます。

ブール型キーワード

- ok - レスポンスステータスコードは 2XX Success でした。
- error - レスポンスステータスコードは 4XX Client Error でした。
- throttle - レスポンスステータスコードは 429 Too Many Requests でした。
- fault - レスポンスステータスコードは 5XX Server Error でした。
- partial - リクエスト内に不完全なセグメントがあります。
- inferred - リクエスト内に推定セグメントがあります。
- first - 要素は列挙リストの最初です。
- last - 要素は列挙リストの最後です。

- `remote` - 根本原因のエンティティがリモートです。
- `root` - サービスはエントリポイント、またはトレースのルートセグメントです。

ブール演算子は、指定されたキーが `true` または `false` のセグメントを見つけます。

ブール演算子

- none - キーワードが true の場合、式は true と評価されます。
- ！- キーワードが false の場合、式は true と評価されます。
- =,!= - キーワードの値を文字列 `true`または`false` と比較します。これらの演算子は、他の演算子と同じように動作しますが、より明示的です。

Example - レスポンスステータスが 2XX OK である

```
ok
```

Example - レスポンスステータスが 2XX OK ではない

```
!ok
```

Example - レスポンスステータスが 2XX OK ではない

```
ok = false
```

Example - 最後の列挙障害トレースにエラー名「逆シリアル化」がある

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

Example - カバレッジが 0.7 より大きく、サービス名が「トレース」であるリモートセグメントを持つリクエスト

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

Example - サービスタイプが「AWS:DynamoDB」の推定セグメントがあるリクエスト

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

Example - ルートとして「data-plane」という名前のセグメントのあるリクエスト

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

数値型キーワード

数値型キーワードを使用して、特定の応答時間、期間、応答ステータスを含むリクエストを検索します。

数値型キーワード

- `responsetime` - サーバーでレスポンスの送信に要した時間。
- `duration` - すべてのダウンストリーム呼び出しを含むリクエスト総所要時間。
- `http.status` - レスポンスステータスコード。
- `index` - 列挙リスト内の要素の位置。
- `coverage` - ルートセグメントの応答時間に対するエンティティの応答時間の 10 進数の割合。応答時間の根本原因のエンティティにのみ適用されます。

数値型演算子

数値型キーワードでは、標準の品質と比較演算子を使用しています。

- =,!= - キーワードが数値と同等か、等しくない。
- <,<=, >,>= - キーワードが数値より小さい、または大きい。

Example - レスポンスステータスが 200 OK ではない

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

Example - 総所要時間が 5〜8 秒のリクエスト

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

Example - すべてのダウンストリーム呼び出しを含めて 3 秒未満で正常に完了したリクエスト

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

Example - 5 より大きいインデックスを持つ列挙型リストエンティティ

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

Example - 最後のエンティティが 0.8 より大きいカバレッジを持つリクエスト

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

文字列型キーワード

文字列型キーワードを使用すると、リクエストヘッダーに特定のテキストを含むトレースや特定のユーザー ID のトレースを見つけることができます。

文字列型キーワード

- `http.url` - リクエストの URL。
- `http.method` - リクエストメソッド。
- `http.useragent` - リクエストのユーザーエージェント文字列。
- `http.clientip` - リクエスタの IP アドレス。
- `user` - Trace の任意の セグメント におけるユーザーフィールドの値。
- `name` - サービスの名前、または例外。
- `type` - サービスタイプ。
- `message` - 例外メッセージ。
- `availabilityzone` - トレース内の任意のセグメントのアベイラビリティーゾーンフィールドの値。
- `instance.id` - トレース内の任意のセグメントのインスタンス ID フィールドの値。
- `resource.arn` - トレース内の任意のセグメントのリソース ARN フィールドの値。

文字列演算子は、特定のテキストと一致する値、または特定のテキストを含む値を見つけます。値は、必ず引用符で囲います。

文字列演算子

- `=,!=` - キーワードが数値と同等か、等しくない。
- `CONTAINS` - キーワードに特定の文字列が含まれている。
- `BEGINSWITH` , `ENDSWITH` - キーワードが特定の文字列で始まる、または終わる。

Example - Http.url フィルター

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

フィールドがトレースに存在するかどうかをテストするには、その値に関係なく、空の文字列が含まれているかどうかを確認します。

Example - ユーザー フィルター

ユーザー ID を持つすべてのトレースを見つけます。

```
user CONTAINS ""
```

Example - 「Auth」という名前のサービスを含む、障害の根本原因を含むトレースの選択

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

Example - 最後のサービスに DynamoDB タイプがある応答時間の根本原因を含むトレースの選択

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

Example - 最後の例外に「account_id:1234567890 のアクセスが拒否されました」というメッセージのある障害の根本原因を含むトレースの選択

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

複合型キーワード

複雑なキーワードを使用し、サービス名、エッジ名、または注釈値に基づいてリクエストを見つけます。サービスとエッジについては、サービスまたはエッジに適用される追加のフィルタ式を指定できます。注釈では、ブール値、数値、または文字列演算子を使用して、特定のキーで注釈の値をフィルタリングできます。

複合型キーワード

- `annotation.`*`key`* - フィールド *`[key]`* (キー)の注釈の値。注釈の値は、ブール値、数値、文字列のいずれかであるため、このタイプの比較演算子のいずれも使用することができます。このキーワードは `service` または `edge` キーワードと組み合わせて使用することができます。

- edge(*source*, *destination*) {*filter*} - *[source]* (ソース)サービスと *[destination]* (宛先) サービス間の接続。オプションの中括弧には、この接続のセグメントに適用されるフィルタ式を含めることができます。
- group.*name* / group.*arn* — グループ名またはグループ ARN で参照されるグループのフィルター式の値。
- json - JSON 根本原因オブジェクト。JSON エンティティをプログラムで作成する手順については、AWS 「X-Ray からデータを取得する」を参照してください。
- service(*name*) {*filter*} - 名前が *[name]* のサービス。オプションの中括弧には、サービスで作成されたセグメントに適用されるフィルタ式を含めることができます。

サービスキーワードを使用して、トレースマップ上の特定のノードにヒットしたリクエストのトレースを検索します。

複合型キーワード演算子は、指定されたキーが設定されている、または設定されていないセグメントを見つけます。

複合型キーワード演算子

- none - キーワードが 設定されている場合、式は true と評価されます。キーワードがブール型の場合は、ブール値として評価されます。
- ! - キーワードが setでない場合、式は true と評価されます。キーワードがブール型の場合は、ブール値として評価されます。
- =,!= — キーワードの値を比較します。
- edge(*source*, *destination*) {*filter*} - *[source]* (ソース) サービスと *[destination]* (宛先) サービス間の接続。オプションの中括弧には、この接続のセグメントに適用されるフィルタ式を含めることができます。
- annotation.*key* - フィールド *[key]* (キー)の注釈の値。注釈の値は、ブール値、数値、文字列のいずれかであるため、このタイプの比較演算子のいずれも使用することができます。このキーワードは service または edge キーワードと組み合わせて使用することができます。
- json - JSON 根本原因オブジェクト。JSON エンティティをプログラムで作成する手順については、AWS 「X-Ray からデータを取得する」を参照してください。

サービスキーワードを使用して、トレースマップ上の特定のノードにヒットしたリクエストのトレースを検索します。

Example - サービスフィルター

障害 (500 シリーズのエラー) が発生した api.example.com の呼び出しを含んだリクエスト。

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

サービス名を除外して、フィルタ式をサービスマップのすべてのノードに適用することができます。

Example - サービスフィルター

トレースマップ上の任意の場所で障害が発生したリクエスト。

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

エッジキーワードは、フィルタ式を 2 つのノード間の接続に適用します。

Example - エッジ フィルター

サービス api.example.com から backend.example.com への呼び出しが失敗してエラーが発生したリクエスト。

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

また、サービスまたはエッジのキーワードを含む！演算子を使用して、サービスまたはエッジを他のフィルタ式の結果から除外することもできます。

Example - サービスおよびリクエスト フィルター

URL が http://api.example.com/ で始まって /v2/ を含むが、api.example.com という名前のサービスに達しないリクエスト。

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

Example — サービスと応答時間のフィルター

http url が設定され、応答時間が 2 秒を超えているトレースを見つけてください。

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

注釈について、annotation.*key* が設定されているか、値のタイプに対応する比較演算子を使用しているかすべてのトレースを呼び出すことができます。

Example - 文字列値を含む注釈

文字列値 gameid の "817DL6V0" という名前のリクエスト。

```
annotation.gameid = "817DL6V0"
```

Example — 注釈が設定されている

age 設定という名前の注釈を持つリクエスト。

```
annotation.age
```

Example — 注釈が設定されていません

age 設定という名前の注釈のないリクエスト。

```
!annotation.age
```

Example - 数値を含む注釈

アノテーション期間が数値 29 より大きいリクエスト。

```
annotation.age > 29
```

Example – サービスまたはエッジと組み合わせた注釈

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

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

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

Example — ユーザーとグループ化

トレースがを満たすリクエストhigh_response_time グループフィルター (例:responseTime > 3)。ユーザーの名前はアリス。

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

Example - 根本原因のエンティティを含む JSON

根本原因エンティティが一致するリクエスト

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

id 関数

service または edge のキーワードにサービス名を入力すると、そのサービス名を含むすべてのノードの結果が得られます。詳細にフィルタリングする場合は、id 関数を使用してサービスのタイプと名前を指定し、同一名を持つノードを区別することができます。

モニタリングアカウント内の複数のアカウントのトレースを表示する場合、account.id 関数を使用してサービスの特定のアカウントを指定します。

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

サービスフィルタおよびエッジフィルタで、サービス名ではなく、id 関数を使用することもできます。

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

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

例えば、 AWS Lambda 関数はトレースマップに 2 つのノードを作成します。1 つは関数の呼び出し用、もう 1 つは Lambda サービス用です。この 2 つのノードは同じ名前ですが、タイプが異なります。標準サービスフィルタは、両ノードのトレースを見つけます。

Example - サービスフィルター

random-name という名前を持つすべてのサービスにエラーを含むリクエスト。

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

id 関数を使用して、関数そのもののエラーを絞り込み、サービスからエラーを除外します。

Example - id 関数を使用したサービスフィルター

サービスタイプが random-name の AWS::Lambda::Function という名前のサービスにエラーを含むリクエスト。

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

タイプ別にノードを検索するには、名前を完全に除外します。

Example – id 関数とサービスタイプを使用したサービスフィルター

サービスタイプが AWS::Lambda::Function のサービスにエラーを含むリクエスト。

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

特定の のノードを検索するには AWS アカウント、アカウント ID を指定します。

Example – id 関数とアカウント ID を使用したサービスフィルター

特定のアカウント ID AWS::Lambda::Function 内のサービスを含むリクエスト。

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

クロスアカウントトレース

AWS X-Ray はクロスアカウントオブザーバビリティ をサポートしているため、 内の複数のアカウントにまたがるアプリケーションをモニタリングおよびトラブルシューティングできます AWS リージョン。リンクされたアカウントのメトリクス、ログ、トレースをまとめてシームレスに検索、可視化、分析できます。これにより、複数のアカウントにまたがるリクエストの全体像を把握できます。クロスアカウントトレースは、CloudWatchコンソール 内の X-Ray トレースマップとトレースページで表示できます。

共有されるオブザーバビリティデータには、次のいずれかのタイプのテレメトリが含まれます。

- Amazon のメトリクス CloudWatch
- Amazon Logs CloudWatch のロググループ
- のトレース AWS X-Ray
- Amazon CloudWatch Application Insights のアプリケーション

クロスアカウントオブザーバビリティの設定

クロスアカウントオブザーバビリティを有効にするには、1 つ以上の AWS モニタリングアカウントを設定し、複数のソースアカウントにリンクさせます。モニタリングアカウントは、ソースアカウントから生成されたオブザーバビリティデータを表示して操作 AWS アカウント できる中心的なアカウントです。ソースアカウントは、含まれ AWS アカウント るリソースのオブザーバビリティデータを生成する個人です。

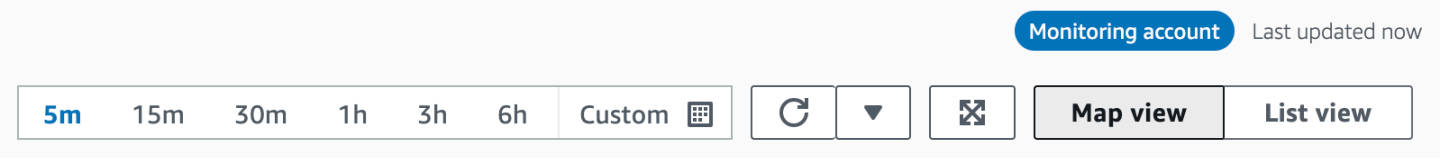
ソースアカウントは、オブザーバビリティデータをモニタリングアカウントと共有します。トレースは各ソースアカウントから最大 5 つのモニタリングアカウントにコピーされます。ソースアカウントから最初のモニタリングアカウントへのトレースのコピーは無料です。追加のモニタリングアカウントに送信されたトレースのコピーは、標準料金に基づいて各ソースアカウントに請求されます。詳細については、「 のAWS X-Ray 料金」および「Amazon の CloudWatch 料金」を参照してください。

モニタリングアカウントとソースアカウント間のリンクを作成するには、 CloudWatch コンソールを使用するか、 AWS CLI および API の新しい Observability Access Manager コマンドを使用します。詳細については、CloudWatch 「クロスアカウントオブザーバビリティ」を参照してください。

> **Note**
>
> X-Ray トレースは、受信した AWS アカウント に請求されます。サンプリングされたリクエストが複数の のサービスにまたがる場合 AWS アカウント、各アカウントは個別のトレースを記録し、すべてのトレースは同じトレース ID を共有します。クロスアカウントオブザーバビリティの料金の詳細については、「 AWS X-Ray の料金」および「Amazon の CloudWatch 料金」を参照してください。

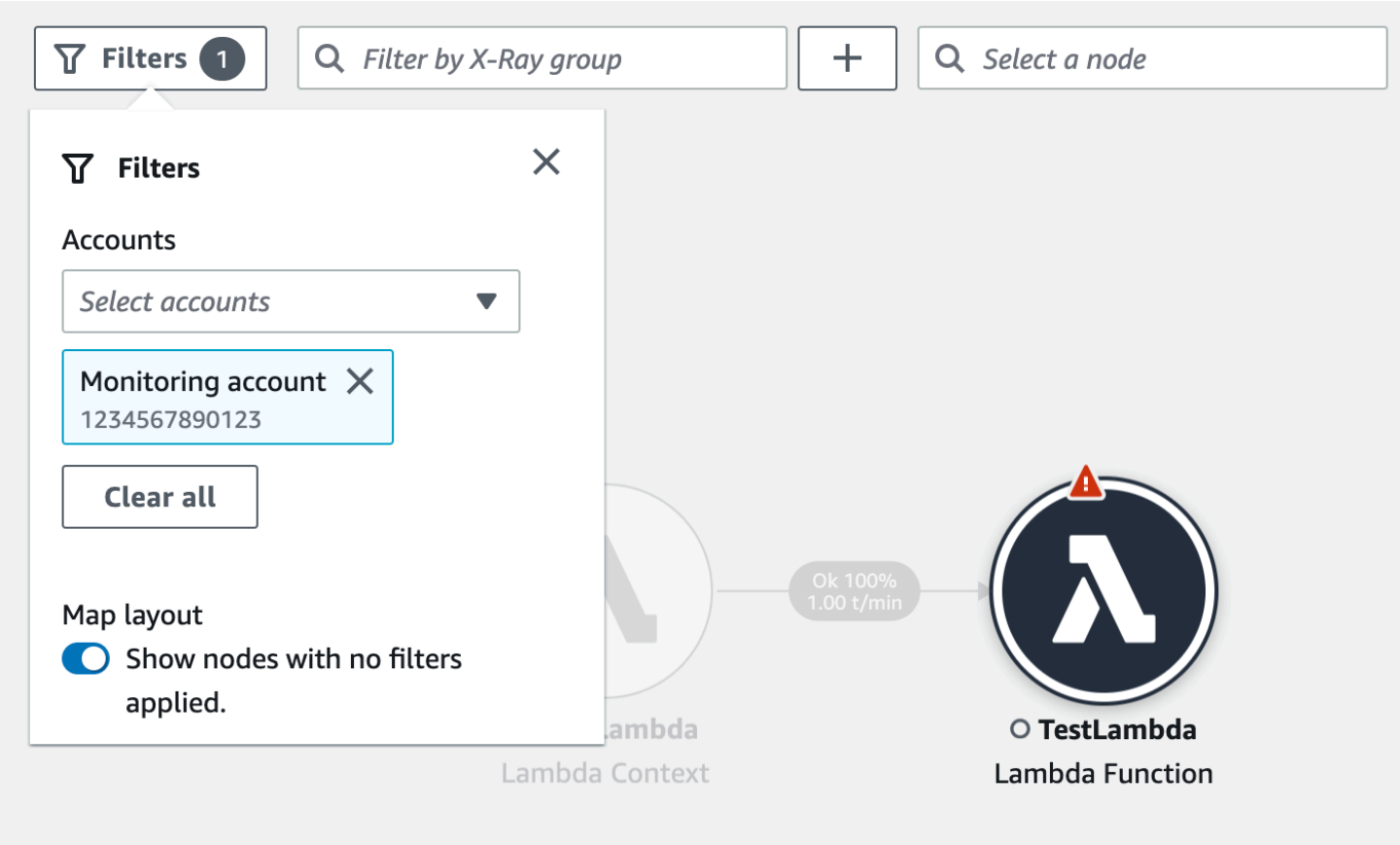
クロスアカウントトレースの表示

クロスアカウントトレースはモニタリングアカウントに表示されます。各ソースアカウントには、その特定のアカウントのローカルトレースのみが表示されます。以下のセクションでは、モニタリングアカウントにサインインし、Amazon CloudWatch コンソールを開いていることを前提としています。トレースマップページとトレースページの両方で、右上隅にモニタリングアカウントバッジが表示されます。

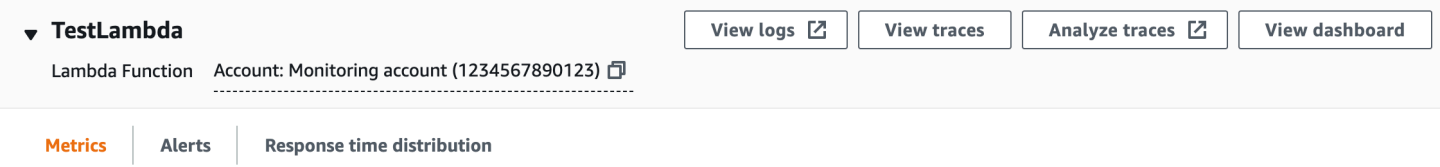


トレースマップ

CloudWatch コンソールで、左側のナビゲーションペインから X-Ray トレースの下にあるトレースマップを選択します。デフォルトでは、トレースマップには、モニタリングアカウントにトレースを送信するすべてのソースアカウントのノードと、モニタリングアカウント自体のノードが表示されます。トレースマップで、左上からフィルターを選択して、Accounts ドロップダウンを使用してトレースマップをフィルタリングします。 アカウントフィルターが適用されると、現在のフィルターと一致しないアカウントのサービスノードはグレー表示されます。

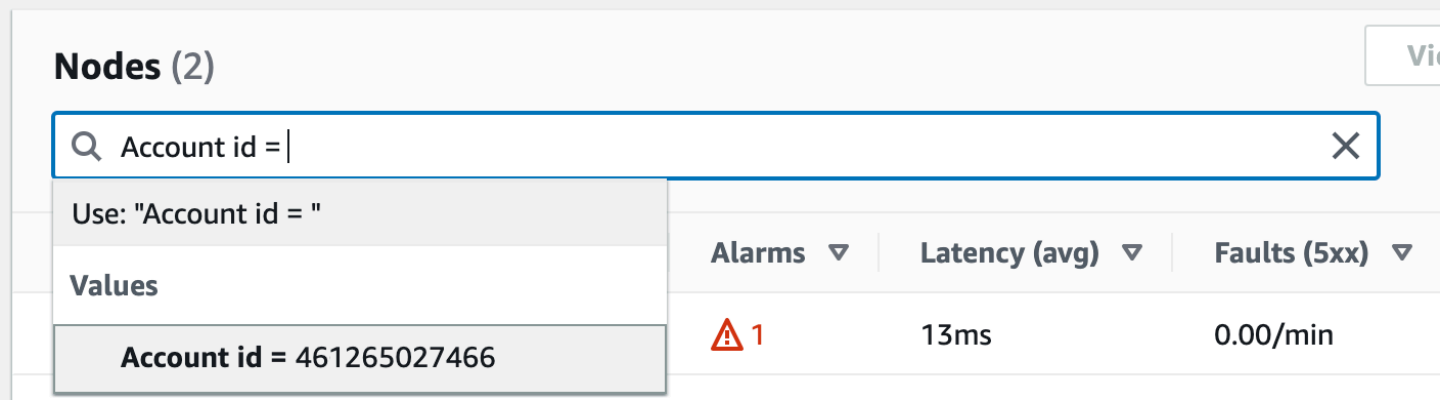


サービスノードを選択すると、[ノードの詳細] ペインにサービスのアカウント ID とラベルが表示されます。



トレースマップの右上隅で、リストビューを選択してサービスノードのリストを表示します。サービスノードのリストには、モニタリングアカウントのサービスと、ソースアカウントに設定されている

すべてのアカウントのサービスが含まれます。[ノード] フィルターで [アカウントラベル] または [アカウント ID] を選択して、ノードのリストをフィルタリングします。

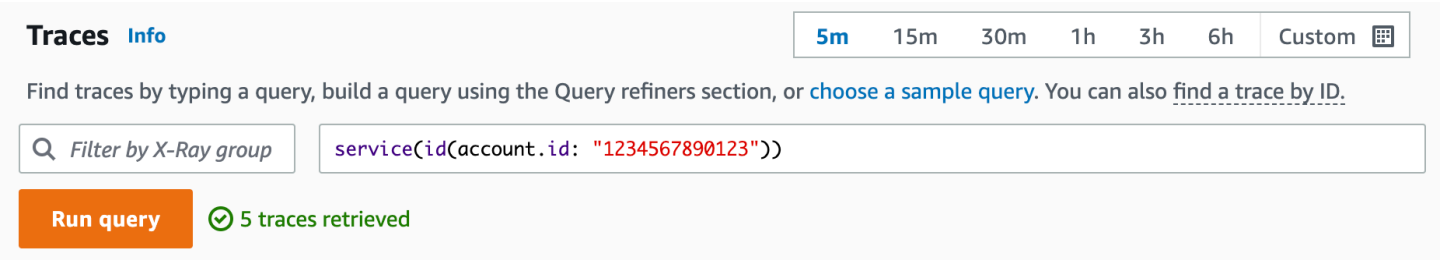


トレース

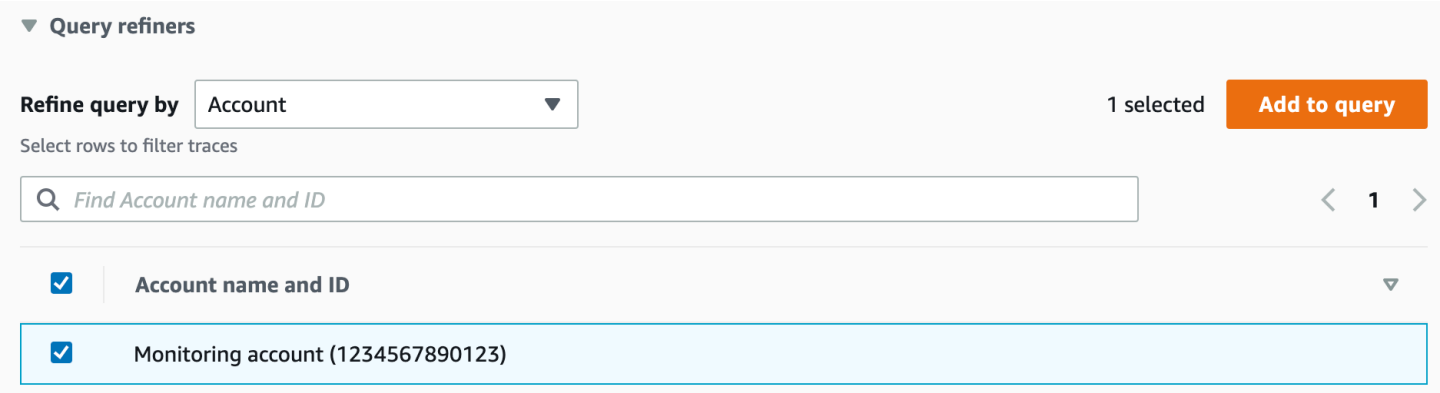
モニタリングアカウントから CloudWatch コンソールを開き、左側のナビゲーションペインの X-Ray トレースでトレースを選択して、複数のアカウントにまたがるトレースの詳細を表示します。このページを開くには、X-Ray トレースマップ でノードを選択し、ノードの詳細ペインからトレースを表示を選択します。

[トレース] ページでは、アカウント ID によるクエリがサポートされています。はじめに、1 つまたは複数のアカウント ID を含むクエリを入力します。クエリの詳細については、「」を参照してくださいフィルター式を使用する。次の例では、アカウント ID Xまたは Y を通過したトレースをクエリします。

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



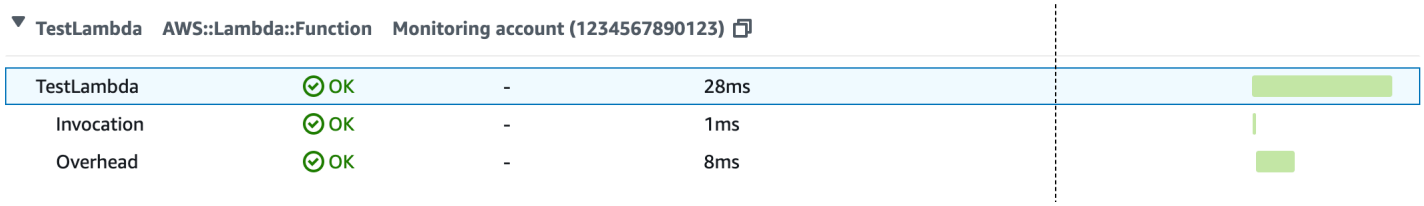
[アカウント] ごとにクエリを絞り込みます。リストから 1 つ以上のアカウントを選択し、[クエリに追加] を選択します。



トレースの詳細

[トレース] ページの下部にある [トレース] リストからトレースを選択すると、トレースの詳細が表示されます。トレースの詳細が表示されます。トレースの詳細マップには、トレースが通過したすべてのアカウントからのサービスノードが含まれます。特定のサービスノードを選択すると、対応するアカウントが表示されます。

[セグメントのタイムライン] セクションには、タイムラインの各セグメントのアカウント詳細が表示されます。



イベント駆動型アプリケーションをトレースする

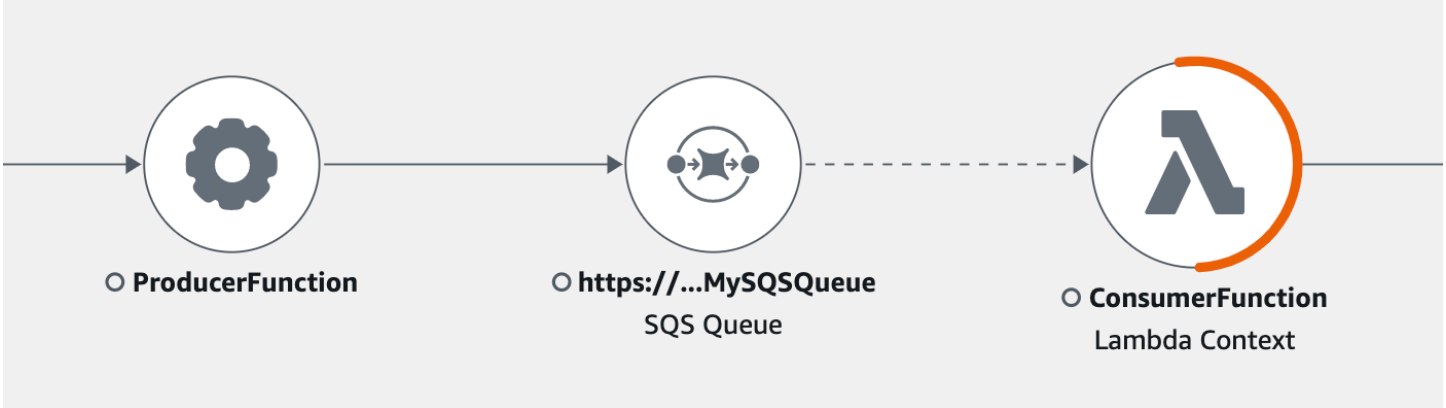
AWS X-Ray は、Amazon SQS および を使用したイベント駆動型アプリケーションのトレースをサポートします AWS Lambda。 CloudWatch コンソールを使用して、Amazon SQS でキューに入れられ、1 つ以上の Lambda 関数によって処理される各リクエストの接続されたビューを表示します。アップストリームメッセージプロデューサーからのトレースは、ダウンストリーム Lambda コンシューマーノードからのトレースに自動的にリンクされ、アプリケーションの end-to-end ビューが作成されます。

> **Note**
>
> 各トレースセグメントは最大 20 のトレースにリンクできます。また、1 つのトレースには最大 100 のリンクを含めることができます。シナリオによっては、追加のトレースをリンクするとトレースドキュメントの最大サイズを超え、トレースが不完全になる可

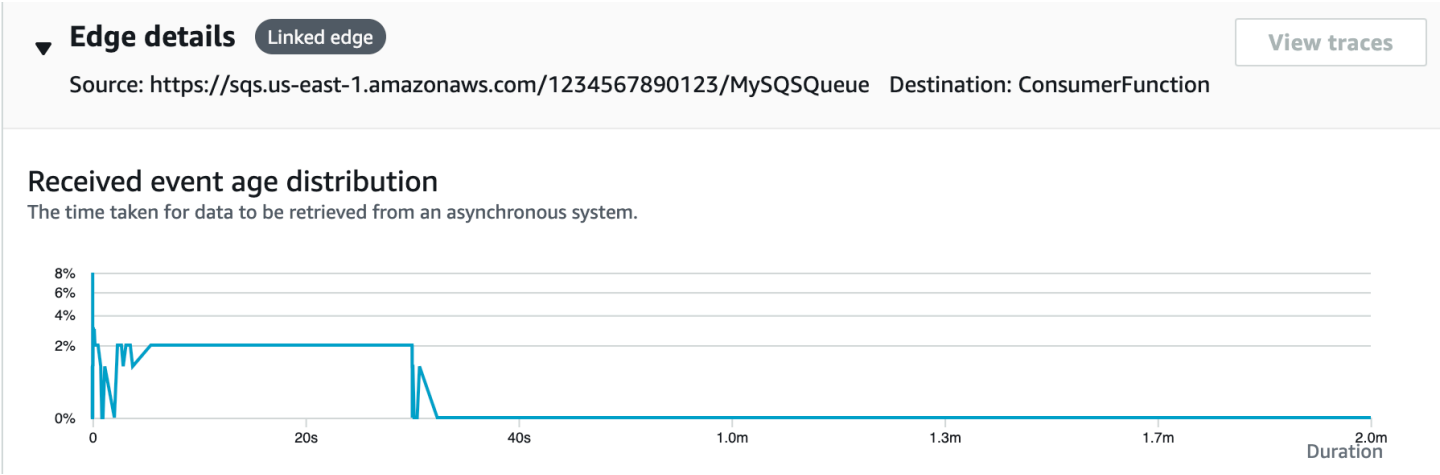
能性があります。例えば、トレーシングが有効になっている Lambda 関数が 1 回の呼び出しで多数の SQS メッセージをキューに送信した場合に発生することがあります。この問題が発生した場合は、X-Ray SDK を使用する緩和策があります。詳細については、Java、Node.js、Python、Go、または .NET 用の X-Ray SDK を参照してください。

リンクされたトレースをトレースマップに表示する

CloudWatch コンソール内のトレースマップページを使用して、Lambda コンシューマーからのトレースにリンクされたメッセージプロデューサーからのトレースを含むトレースマップを表示します。これらのリンクは、Amazon SQS ノードとダウンストリーム Lambda コンシューマーノードを接続する破線のエッジで表示されます。



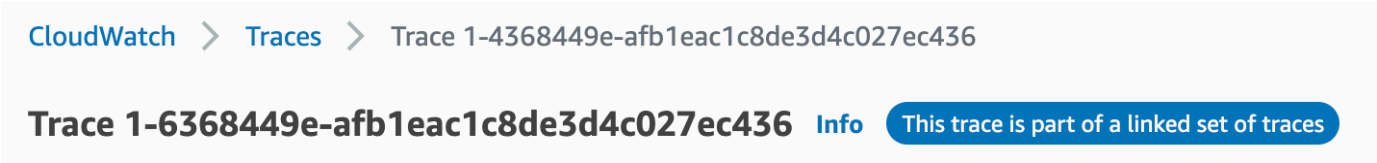
破線のエッジを選択すると、[受け取ったイベントが発生してから経過した時間]ヒストグラムが表示されます。これは、コンシューマーが受け取ったイベントの経過時間の分布をマッピングしたものです。経過時間は、イベントを受信するたびに計算されます。



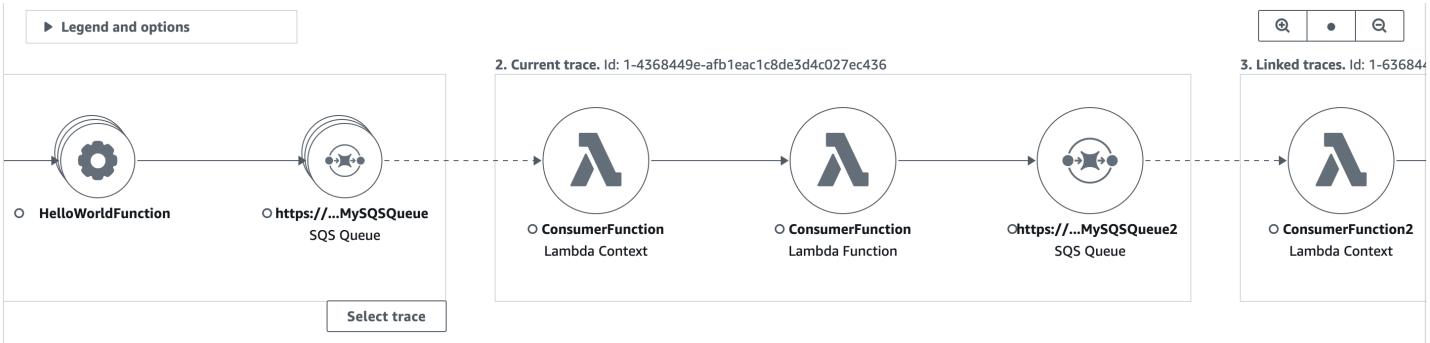
リンクされたトレースの詳細の表示

メッセージプロデューサー、Amazon SQS キュー、または Lambda コンシューマーから送信されたトレースの詳細を表示します。

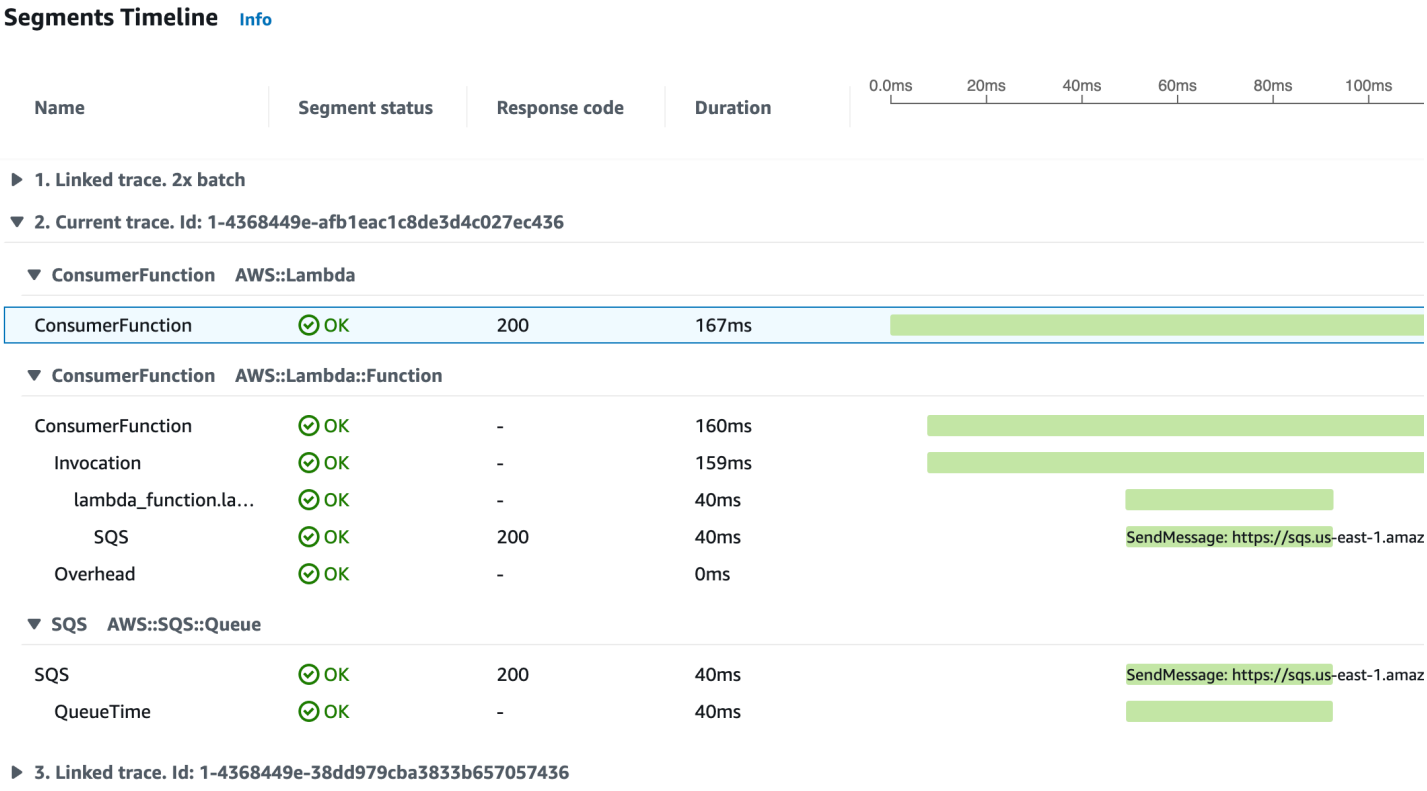
1. トレースマップを使用して、メッセージプロデューサー、Amazon SQS、または Lambda コンシューマーノードを選択します。
2. ノードの詳細ペインから [トレースを表示] を選択すると、トレースのリストが表示されます。CloudWatch コンソール内のトレースページに直接移動することもできます。
3. リストから特定のトレースを選択し、トレースの詳細ページを開きます。選択したトレースが、リンクされたトレースセットの一部である場合、トレースの詳細ページにメッセージが表示されます。



トレースの詳細マップには、現在のトレースと、アップストリームおよびダウンストリームにリンクされたトレースが表示されます。各トレースは、各トレースの境界を示すボックス内に含まれています。現在選択されているトレースがアップストリームまたはダウンストリームの複数のトレースにリンクされている場合、アップストリームまたはダウンストリームにリンクされているトレース内のノードは積み重ねられ、[トレースを選択] ボタンが表示されます。



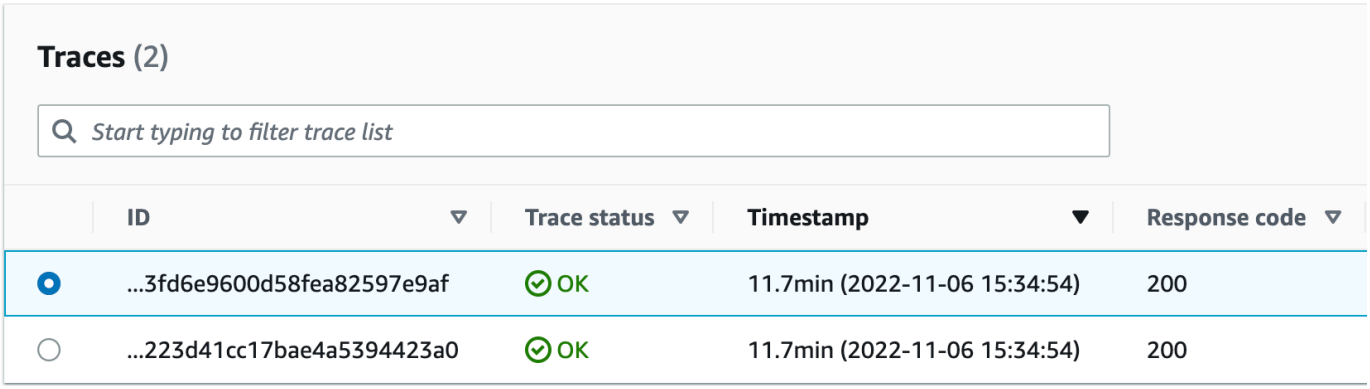
トレース詳細マップの下に、アップストリームとダウンストリームにリンクされたトレースを含むトレースセグメントのタイムラインが表示されます。アップストリームまたはダウンストリームにリンクされたトレースが複数ある場合、それらのセグメントの詳細は表示できません。リンクされたトレースのセット内の 1 つのトレースのセグメントの詳細を表示するには、次のセクションで説明するように 1 つのトレースを選択します。



リンクされたトレースのセットから 1 つのトレースを選択

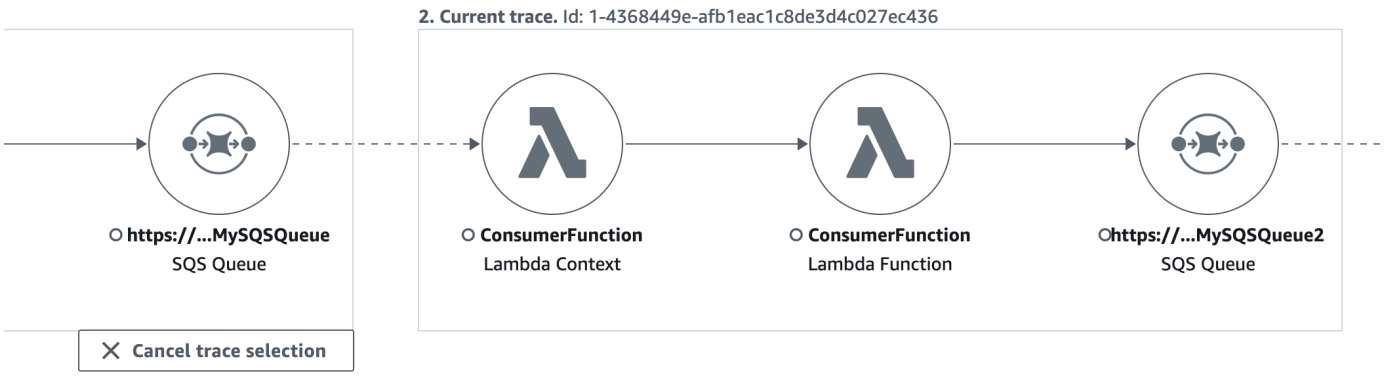
リンクされたトレースセットを 1 つのトレースにフィルタリングすると、タイムラインにセグメントの詳細が表示されます。

1. トレース詳細マップのリンクされたトレースの下にあるトレースを選択を選択します。トレースのリストが表示されます。



2. トレースの横にあるラジオボタンを選択すると、トレースの詳細マップ内でそのトレースが表示されます。

3. [トレースの選択をキャンセル] を選択すると、リンクされたトレースのセット全体が表示されます。



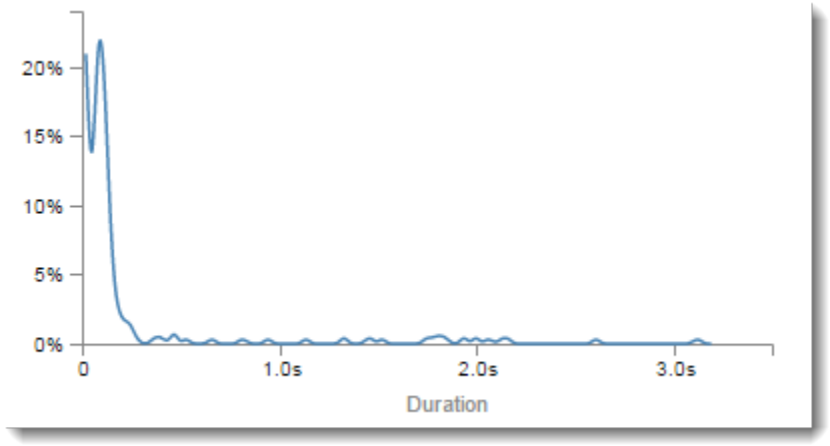
レイテンシーヒストグラムを使用する

トレースマップでノードまたはエッジを選択すると、X-Ray コンソールにレイテンシー分布ヒストグラムが表示されます。

レイテンシー

レイテンシーは、リクエストが開始してから完了するまでの時間です。ヒストグラムは、レイテンシーの分散を示します。また、期間を x 軸、各期間に一致するリクエストの割合を y 軸に示しています。

このヒストグラムでは、ほとんどのリクエストを 300 ミリ秒 (ms) 以下で完了するサービスを示します。割合が低いリクエストには最大 2 秒、異常値の場合はそれ以上かかります。



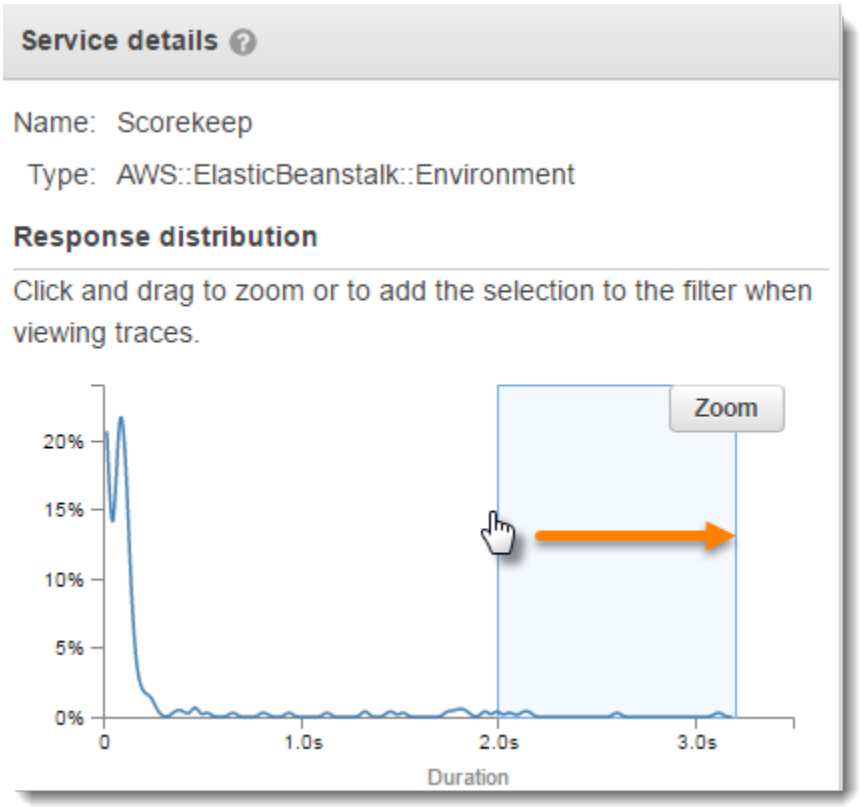
サービスの詳細の解釈

サービス ヒストグラムおよびエッジヒストグラムは、レイテンシーをサービスまたはリクエスト元の視点からビジュアルに表現したものです。

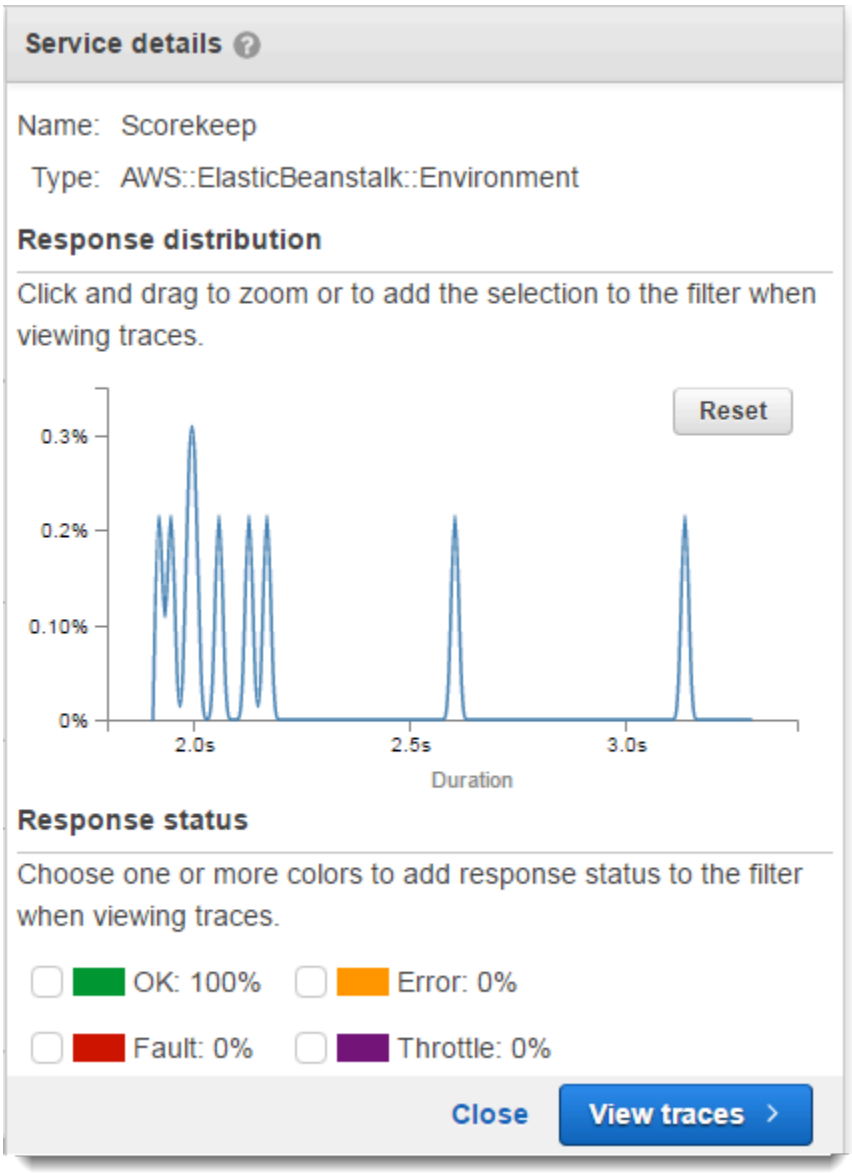
- 円をクリックして、[service node] (サービス ノード)を選択します。X-Ray は、サービスによる依頼者のヒストグラムを示します。このレイテンシーは、サービスによって記録されたものであり、サービスと依頼者の間のネットワークレイテンシーは含まれません。
- 2 つのサービスの間のエッジの線、または矢の先をクリックして、[edge] (エッジ) を選択します。X-Rayは、ダウンストリームサービスによって行われる依頼者のリクエストのヒストグラムを示します。このレイテンシーは、依頼者によって記録されたものであり、2 つのサービスの間のネットワーク接続のレイテンシーは含まれません。

[サービスの詳細] パネルヒストグラムを解釈するには、ヒストグラムの値の大部分と異なる値を探します。このような 異常値 は、ヒストグラムのピークまたは急増としてみなすことができるため、特定のエリアのトレースを表示して、現在の状況を調査することができます。

レイテンシーでフィルタリングされたトレースを表示するには、ヒストグラムの範囲を選択します。選択の開始位置をクリックし、左から右にドラッグして、トレースフィルタに含むレイテンシーの範囲をハイライト表示します。



範囲を選択したら、[Zoom] を選択してヒストグラムの部分のみを表示し、選択範囲を絞り込みます。



表示するエリアを絞って設定したら、[View traces] を選択します。

X-Ray Insights を使用する

AWS X-Ray は、アカウント内のトレースデータを継続的に分析して、アプリケーションの緊急の問題を特定します。故障率が想定範囲を超えた場合、その問題を記録し、解決されるまで影響を追跡するインサイトを作成します。Insights を使用すると、次のことが可能になります。

- アプリケーションで問題が発生している場所、問題の根本原因、および関連する影響を特定します。 Insights によって提供される影響分析により、問題の重要度と優先度を導き出すことができます。
- 時間の経過とともに課題が変化すると、通知を受信します。 Insights 通知は、Amazon を使用してモニタリングおよびアラートソリューションと統合できます EventBridge。この統合により、問題の重要度に基づいて自動メールまたはアラートを送信できます。

X-Ray コンソールは、トレースマップで進行中のインシデントがあるノードを識別します。インサイトの概要を確認するには、影響を受けるノードを選択します。左側のナビゲーションペインから Insights を選択して、Insights を表示およびフィルタリングすることもできます。

X-Ray は、サービスマップの 1 つ以上のノードで[anomaly] (異常)を検出すると、インサイトを作成します。このサービスは、統計モデリングを使用して、アプリケーション内のサービスの予想故障率を予測します。前の例では、異常は からの障害の増加です AWS Elastic Beanstalk。Elastic Beanstalk サーバーで複数の API コール タイムアウトが発生し、ダウンストリームノードで異常が発生しました。

X-Ray コンソールで Insights を有効にする

インサイト機能を使用する場合は、グループごとに、インサイトを有効にする必要があります。インサイトを有効にするには、[Groups] (グループ)ページから行います。

1. [X-Ray console] (X-Ray コンソール)を開きます。
2. 既存のグループを選択するか、[Create group] (グループの作成)を選択して新しいグループを作成し、[Enable Insights] (インサイトを有効にする)を選択します。X-Ray コンソールでのグループの構成の詳細については、グループを設定する を参照してください。
3. 左側のナビゲーションペインで、[Insights] (インサイト)を選択し、表示するインサイトを選択します。

> **Note**
>
> X-Ray は GetInsightSummaries、、 GetInsight GetInsightEvents、および GetInsightImpactGraph API オペレーションを使用してインサイトからデータを取得します。インサイトを表示するには、 AWSXrayReadOnlyAccess IAM 管理ポリシーを使用するか、次のカスタムポリシーを IAM ロールに追加します。

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

詳細については、「が IAM と AWS X-Ray 連携する方法」を参照してください。

インサイトの通知を有効にします。

インサイト通知を使用すると、インサイトが作成されたとき、大幅な変更、クローズされたときなど、インサイトイベントごとに通知が作成されます。　お客様は Amazon EventBridge イベントを通じてこれらの通知を受信し、条件付きルールを使用して SNS 通知、Lambda 呼び出し、SQS キューへのメッセージの投稿、または がサポートするターゲット EventBridgeなどのアクションを実行できます。インサイト 通知がベストで出され -ベストエフォートですが、保証されていません。ターゲットの詳細については、「Amazon EventBridge Targets」を参照してください。

インサイトが有効なグループに対してのインサイト通知は、[Groups] (グループ)ページから有効にすることができます。

X-Ray グループの通知を有効にするには

1. [X-Ray console].(X-Ray コンソール)を開きます。
2. 既存のグループを選択するか、[Create group] グループの作成)を選択して新しいグループを作成し、[Enable Insights] (インサイトを有効にする)が選択されていることを確認し、[Enable Notifications] (通知を有効にする)を選択します。X-Ray コンソールでのグループの設定の詳細については、グループを設定する を参照してください。

Amazon EventBridge の条件付きルールを設定するには

1. Amazon EventBridge コンソール を開きます。
2. 左側のナビゲーションバーで、[Rules] (ルール) に移動し[Create rule] (ルールの作成) を選択します。
3. ルールの名前と説明を入力します。
4. [Event pattern] (イベントパターン)を選択してから、[Custom pattern] (カスタムパターン)を選択します。`"source": [ "aws.xray" ]` とならびに `"detail-type": [ "AWS X-Ray Insight Update" ]` を含むパターンを指定します。考えられるパターンとしては、以下のようなものがあります。
   - X-Ray インサイト からのすべての受信イベントを照合するイベントパターン:

```
{
```

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

- 指定した **state** ならびに **category** に一致するイベントパターン:

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

5. イベントがこのルールに一致したときに呼び出すターゲットを選択して設定します。
6. （オプション）このルールをより簡単に識別して選択するためのタグを指定します。
7. [Create] を選択します。

> **Note**
>
> X-Ray インサイト通知は EventBridge、現在カスタマーマネージドキーをサポートしていない Amazon にイベントを送信します。詳細については、「AWS X-Ray でのデータ保護」を参照してください。

インサイト 概要

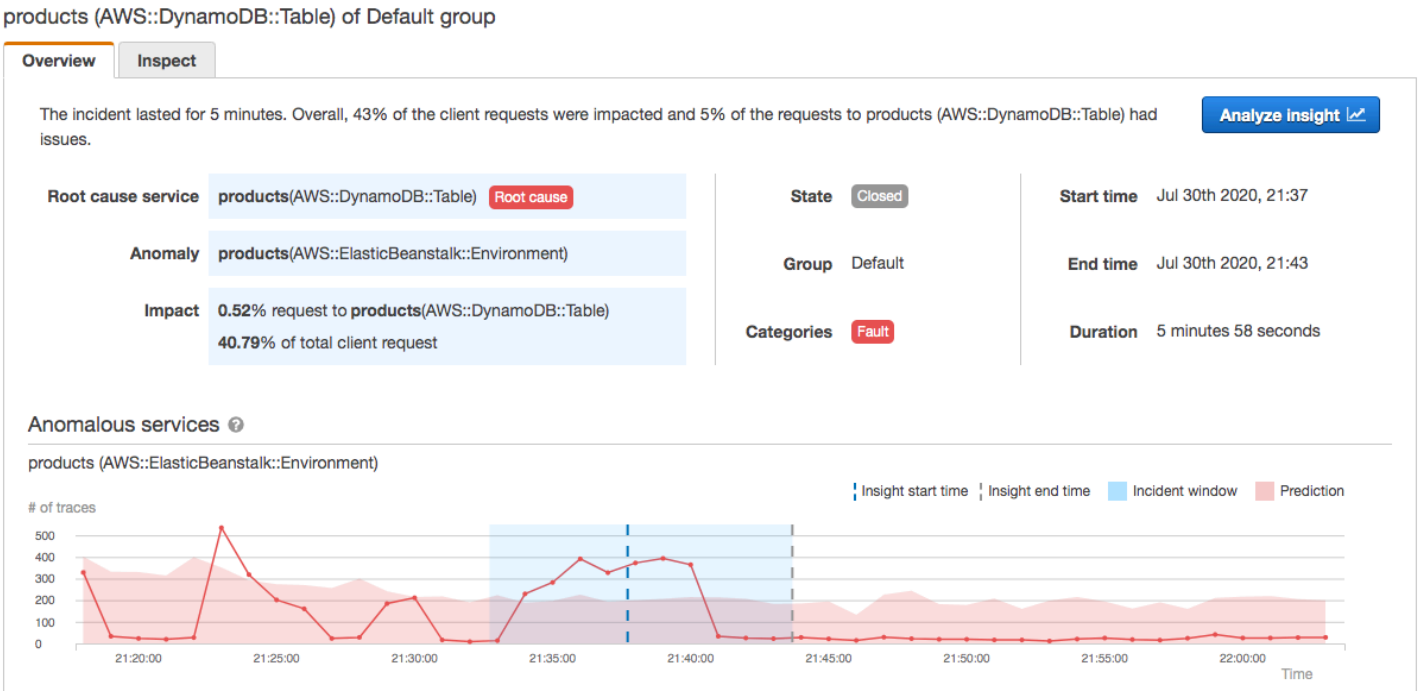
インサイト 試行tの概要ページでは、三つの重要な質問に答えます。

- 根本的な問題は何ですか？
- 根本原因は何ですか？
- その影響は何ですか？

[Anomalous services] (異常サービス)セクションには、インシデント中の故障率の変化を示す各サービスのタイムラインが表示されます。タイムラインには、記録されたトラフィック量に基づいて予想される障害数を示すソリッドバンドに障害が発生したトレース数を重ねて表示されます。インサイ

トの期間は、[Incident window] (インシデント ウインドウ)で表示されます。インシデントウィンドウは、X-Ray がメトリックの異常を観測したときに始まり、インサイトがアクティブである間、継続します。

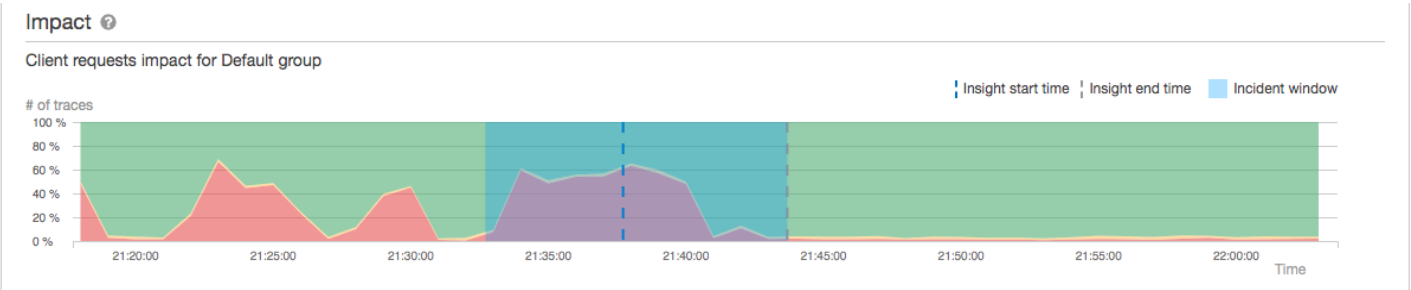
次の例は、インシデントの原因となった障害の増加を示しています。



根本原因セクションには、根本原因サービスと影響を受けるパスに焦点を当てたトレースマップが表示されます。根本原因マップの右上にある目のアイコンを選択すると、影響を受けないノードを非表示にすることができます。　根本原因サービスは、X-Ray が異常を特定した最も遠いダウンストリームノードです。　これは、インストルメントしたサービス、またはサービスがインストルメントクライアントで呼び出した外部サービスを表すことができます。　例えば、計測された AWS SDK クライアントで Amazon DynamoDB を呼び出すと、DynamoDB からの障害が増加すると、根本原因として DynamoDB に関するインサイトが得られます。

根本原因をさらに調査するには、根本原因グラフの [View root cause details] (根本原因の詳細を表示)を選択します。[Analytics] (分析)ページを使用して、根本原因と関連メッセージを調査することができます。詳細については、「Analytics コンソールを操作する」を参照してください。

マップ内で続くアップストリームの障害は、複数のノードに影響し、複数の異常を引き起こす可能性があります。リクエストを行ったユーザーに障害を引き渡された場合、結果は[client fault] (クライアント障害)となります。これはトレースマップのルートノードの障害です。[Impact] (影響) グラフは、グループ全体のクライアント体験のタイムラインを表したものです。この経験は、次の状態のパーセンテージに基づいて計算されます。Fault, Error, Throttle, および Okay。



この例では、インシデントの発生時にルートノードで障害が発生したトレースの増加を示します。ダウンストリームサービスのインシデントは、クライアントエラーの増加に必ずしも対応しているとは限りません。

[Analyze insight] (インサイトを分析)を選択すると X-Ray Analytics コンソールがウィンドウで開き、インサイトの原因となる一連のトレースを深く掘り下げることができます。詳細については、「Analytics コンソールを操作する」を参照してください。

[Understanding impact] (影響の把握)

AWS X-Ray は、インサイトと通知の生成の一環として、進行中の問題によって引き起こされる影響を測定します。影響は、次の 2 つの方法で測定されます。

- X-Ray グループへの影響。詳細については、「グループの設定」を参照してください。
- 根本原因サービスへの影響

この影響は、一定の期間内に失敗またはエラーを引き起こしているリクエストの割合によって決まります。この影響分析では、特定のシナリオに基づいて、問題の重要度と優先度を導き出すことができます。この影響は、インサイト通知に加えて、コンソール体験の一部として利用できます。

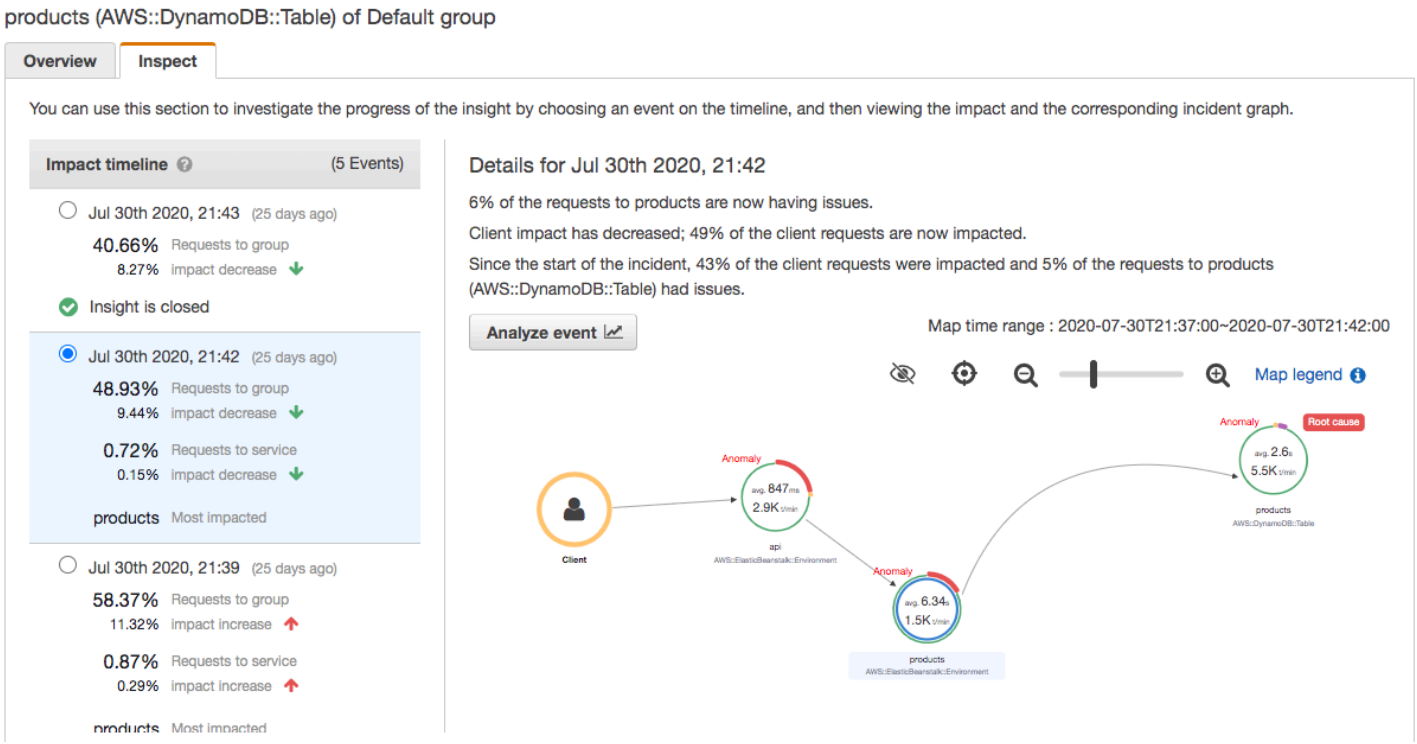
[Deduplication] (重複排除)

AWS X-Ray Insights は、複数のマイクロサービス間で問題を重複排除します。異常検出により、問題の根本原因であるサービスを特定し、他の関連サービスが同じ根本原因で異常な動作を示しているかどうかを判断し、結果を単一のインサイトとして記録します。

インサイトの進捗状況を確認します

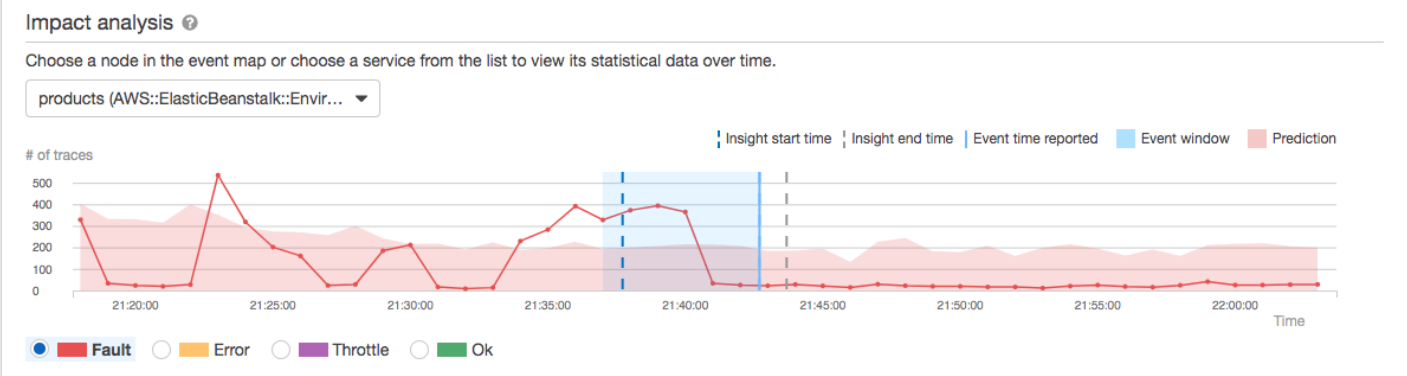
X-Ray は、インサイトが解決されるまで定期的に再評価し、注目すべき中間変更をそれぞれ通知として記録し、Amazon EventBridge イベントとして送信できます。これにより、問題が時間の経過とともにどのように変化したかを判断するためのプロセスとワークフローを構築し、 を使用して E メールの送信やアラートシステムとの統合などの適切なアクションを実行できます EventBridge。

インシデントイベントは、[Inspect] (検査) ページの [Impact Timeline] (影響タイムライン) で確認するすことができます。デフォルトでは、別のサービスを選択するまで、タイムラインには最も影響のあるサービスが表示されます。



イベントのトレースマップとグラフを表示するには、影響タイムラインから選択します。トレースマップには、インシデントの影響を受けるアプリケーション内のサービスが表示されます。[Impact

analysis] (影響分析)では、選択したノードおよびグループ内のクライアントの障害タイムラインがグラフで表示されます。



インシデントに関連するトレースを詳しく調べるには、[Inspect] (検査) ページで [Analyze event] (イベントの分析) を選択します。[Analytics] (分析) ページで、トレースのリストを絞り込み、影響を受けるユーザーを特定することができます。詳細については、「Analytics コンソールを操作する」を参照してください。

Analytics コンソールを操作する

Analytics AWS X-Ray コンソールは、トレースデータを解釈して、アプリケーションとその基盤となるサービスのパフォーマンスをすばやく理解するためのインタラクティブなツールです。このコンソールを使用すると、インタラクティブな応答時間グラフと時系列グラフを使用して、トレースを調査、分析、および視覚化できます。

Analytics コンソールで選択すると、コンソールは選択したすべてのトレースのサブセットを反映するようにフィルタを作成します。現在のトレースセットに関連付けられているグラフとメトリクスおよびフィールドのパネルをクリックして、アクティブなデータセットをきめ細かく絞り込むことができます。

コンソールの機能

X-Ray Analytics コンソールでは、以下の主要な機能を使用して、トレースデータをグループ化、フィルタリング、比較、定量化します。

機能

機能	説明
Groups (グループ)	最初に選択されるグループは、`Default` です。取得されたグループを変更するには、メイ

機能	説明
	ンフィルタ式検索バーの右側にあるメニューから別のグループを選択します。グループの詳細については、「グループの設定」を参照してください。
Retrieved traces (取得されたトレース)	デフォルトでは、Analytics コンソールは選択したグループのすべてのトレースに基づいてグラフを生成します。取得されたトレースは、作業セットのすべてのトレースを表します。このタイルにトレースカウントがあります。メイン検索バーに適用したフィルタ式は、取得されたトレースを絞り込んで更新します。
Show in charts/Hide from charts (グラフに表示/グラフに非表示)	アクティブなグループを切り替え、取得されたトレースと比較します。グループに関連するデータをアクティブなフィルタと比較するには、[Show in charts (グラフに表示)]を選択します。グラフからこのビューを削除するには、[Hide from charts (グラフに非表示)] を選択します。
Filtered trace set A (フィルタリングされたトレースセット A)	グラフやテーブルとの対話を通じて、フィルターを適用して[Filtered trace set A] (フィルタリングされたトレースセット A)の基準を作成します。フィルターが適用されると、適用可能なトレースの数と取得された合計に対するトレースの割合がこのタイル内で計算されます。フィルタは [Filtered trace set A] タイル内のタグとして入力され、タイルから削除することもできます。

機能	説明
Refine (絞り込み)	この関数は、トレースセット A に適用されたフィルタに基づいて、取得したトレースのセットを更新します。取得されたトレースセットを絞り込むと、トレースセット A のフィルタに基づいて取得したすべてのトレースの処理中のセットが更新されます。取得されたトレースのワーキングセットは、グループ内のすべてのトレースをサンプリングしたサブセットです。
Filtered trace set B (フィルタリングされたトレースセット B)	作成した場合、[フィルタリングされたトレースセット B] は、[フィルタリングされたトレースセット A] の写しです。2 つのトレースセットを比較するには、トレースセット A を固定したままトレースセット B に適用する新しいフィルターを選択します。フィルタが適用されると、適用可能なトレースの数と取得された合計からのトレースの割合がこのタイル内で計算されます。フィルタは、[Filtered trace set B] タイル内にタグとして入力され、タイルから削除することもできます。
Response time root cause entity paths (応答時間根本原因エンティティパス)	記録されたエンティティパスのテーブル。X-Rayは、トレース内のどのパスが応答時間の最大の原因であるかを判別します。この形式は、検出されたエンティティの階層を示し、最後に応答時間の根本原因を示します。これらの行を使用して、定期的な応答時間障害をフィルタリングします。根本原因フィルターのカスタマイズと API を介したデータの取得の詳細については、「」の「根本原因分析の取得と改良」セクションを参照してくださいX-Ray からデータを取得する。

機能	説明
Delta (Δ) (デルタ)	トレースセット A とトレースセット B の両方がアクティブなときにメトリクステーブルに追加される列。Deltaデルタ列は、トレースセット A とトレースセット B の間のトレースの割合の差を計算します。

応答時間ディストリビューション

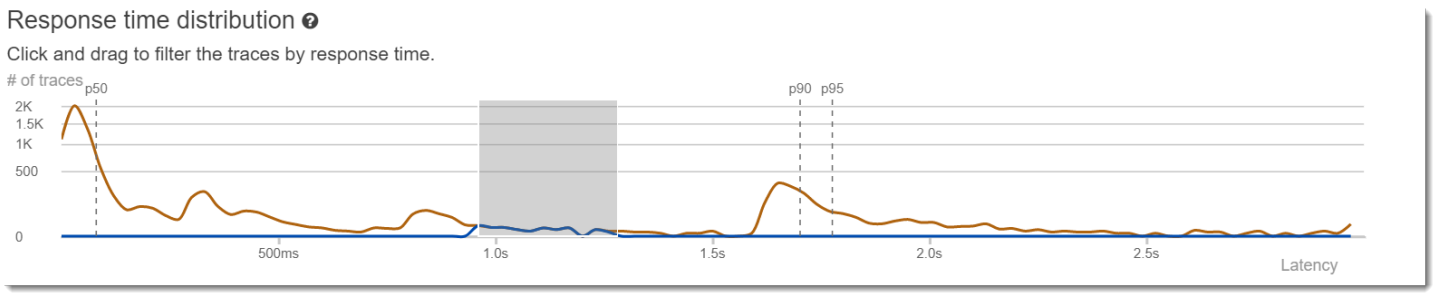
X-Ray Analytics コンソールは、トレースの視覚化に役立つ、[Response Time Distribution] (応答時間ディストリビューション) および [Time Series Activity] (時系列アクティビティ) の 2 つの主要なグラフが生成されます。このセクションと続く部分ではそれぞれの例をあげて、グラフを読み取る方法の基本について説明します。

応答時系列グラフに関連する色は次のとおりです (時系列グラフは同じカラースキームを使用します)。

- [All traces in the group] (グループのすべてのトレース) - グレー
- [Retrieved traces] (取得されたトレース) - オレンジ
- [Filtered trace set A] (フィルタリングされたトレースセット A) - 緑
- [Filtered trace set B] (フィルタリングされたトレースセット B) - 青

Example - 応答時間ディストリビューション

応答時間ディストリビューションは、指定された応答時間でのトレース数を示すグラフです。クリックしてドラッグし、応答時間ディストリビューションで選択を行います。これは、特定の応答時間内のすべてのトレースに対して、`responseTime`という名前の作業トレースセットにフィルターを選択して作成します。

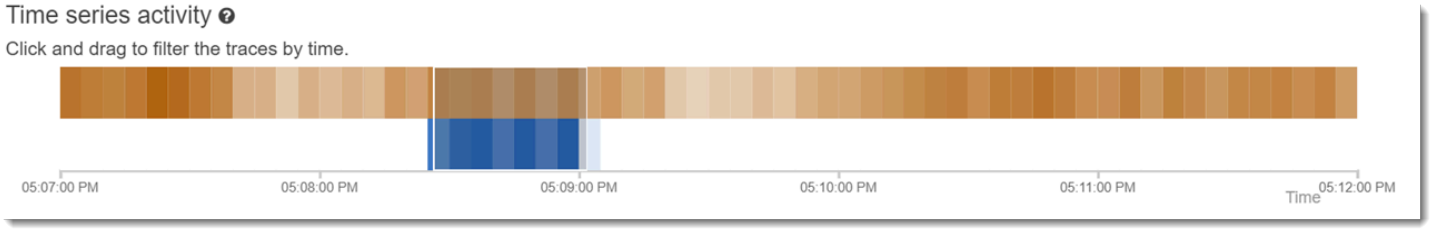


時系列アクティビティ

時系列アクティビティグラフには、指定された期間のトレースの数が表示されます。カラーインジケータは、応答時間ディストリビューションの折れ線グラフの色を反映しています。アクティビティシリーズのカラーブロックが濃く、密になるほど、より多くのトレースが指定された期間に表示されます。

Example - 時系列アクティビティ

クリックしてドラッグし、時系列アクティビティグラフ内で選択を行います。これは、特定の時間範囲内のすべてのトレースに対して、作業トレースセットに指定されている `timerange` という名前のフィルタを選択して作成します。



ワークフローの例

以下の例では、X-Ray Analytics コンソールの一般的なユースケースを示します。各例では、コンソールエクスペリエンスの主要な機能を示します。グループとして、例では基本的なトラブルシューティングワークフローに従います。このステップでは、最初に異常なノードを特定し、次に Analytics コンソールを操作して比較クエリを自動的に生成する方法を示します。クエリによってスコープを絞り込んだら、最終的に関心のあるトレースの詳細を調べて、サービスの健全性を損なう原因を特定します。

サービスグラフの障害を確認する

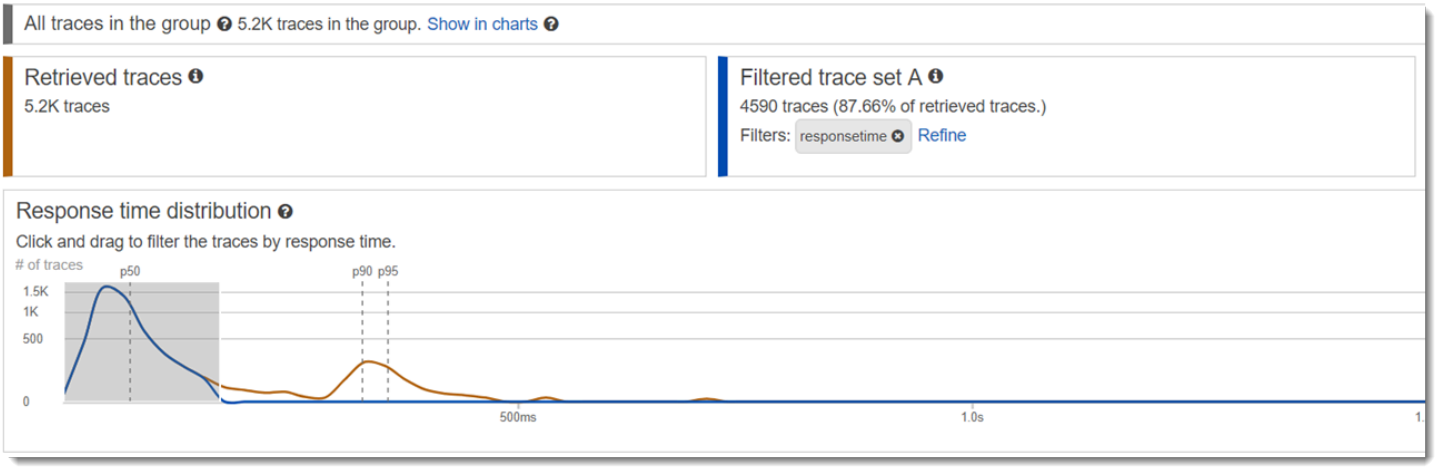
トレースマップは、エラーと障害に対する成功した呼び出しの比率に基づいて色付けすることで、各ノードの状態を示します。赤色の割合が表示されたら、そのシグナルノードに障害が発生しています。X-Ray Analytics コンソールを使用して、調査を実施します。

トレースマップの読み方の詳細については、「X-Ray トレースマップの使用」を参照してください。

ピーク応答時間を特定する

応答時間ディストリビューションを使用して、応答時間のピークを確認できます。応答時間のthe ピークを選択すると、グラフの下のテーブルが更新され、ステータスコードなど、関連付けられているすべてのメトリクスが表示されます。

クリックしてドラッグすると、X-Rayによってフィルターが選択され、作成されます。これは、グラフ化された線の上にグレーの影で表示されます。影をディストリビューションに沿って左右にドラッグして選択内容とフィルタを更新できるようになりました。



ステータスコードでマークされているすべてのトレースを表示する

グラフの下にあるメトリクステーブルを使用して、選択したピーク内のトレースを詳しく調べることができます。[HTTP STATUS CODE] テーブルの行をクリックすると、作業データセットにフィルタが自動的に作成されます。たとえば、ステータスコード 500 のトレースをすべて表示することができます。これにより `http.status` という名前のトレースセットタイルにフィルタタグが作成されます。

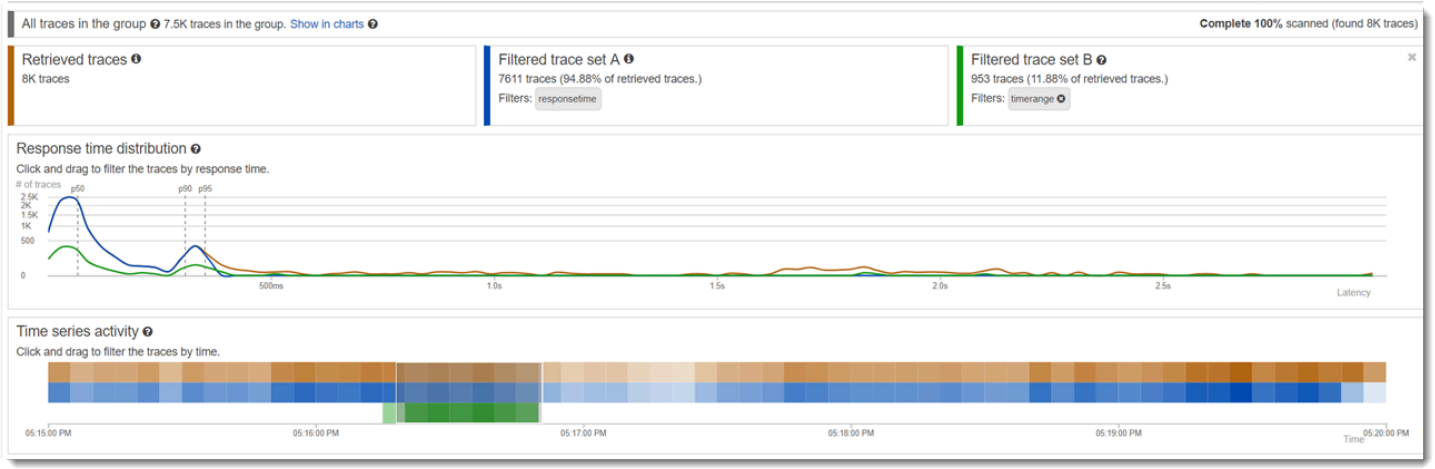
サブグループ内のすべての項目と、ユーザーに関連付けられているすべての項目を表示する

ユーザー、URL、応答時間の根本原因、またはその他の事前定義された属性に基づいてエラーセットを詳しく調べます。たとえば、500 ステータスコードでトレースのセットをさらにフィルタリングするには、[USERS] テーブルから行を選択します。これにより、トレースセットタイルに以前に指定された `http.status`、および `user` の 2 つのフィルタタグが作成されます。

異なる条件のあるトレースのセット 2 つを比較する

さまざまなユーザーとその POST リクエストを比較して、他の不一致や相関関係を見つけます。最初のフィルタのセットを適用します。これらは応答時間ディストリビューションの青い線で定義されます。次に、[Compare (比較)] を選択します。最初に、これによりトレースセット A にフィルタのコピーが作成されます。

続行するには、トレースセット B に適用する新しいフィルタセットを定義します。この 2 番目のセットは緑色の線で表されます。次の例は、青と緑のカラースキームに従って異なる線を示しています。



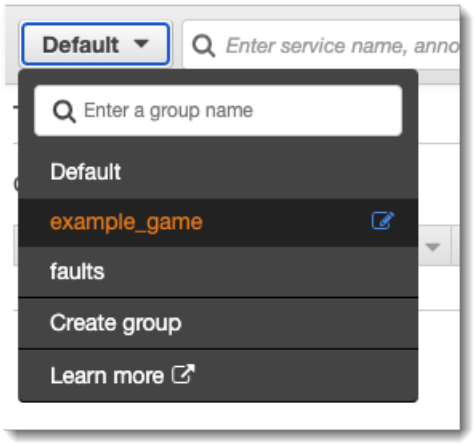
目的のトレースを特定して、詳細を表示する

コンソールフィルタを使用してスコープを絞り込むと、メトリクステーブルの下のトレースリストがより有益になります。トレースリストテーブルは、[URL]、[USER]、および [STATUS CODE] に関する情報を 1 つのビューにまとめたものです。詳細については、このテーブルから行を選択して、トレースの詳細ページを開き、タイムライン、raw データを表示します。

グループを設定する

グループは、フィルタ式で定義されるトレースのコレクションです。グループを使用して、追加のサービスグラフを生成し、Amazon CloudWatch メトリクスを指定できます。 AWS X-Ray コンソールまたは X-Ray API を使用して、サービスのグループを作成して管理できます。このトピックでは、X-Ray コンソールを使用してグループを作成および管理する方法について説明します。X-Ray API を使用してグループを管理する方法については、X-Ray API を使用したサンプリング、グループ、および暗号化設定の構成 を参照してください。

トレースマップ、トレース、または分析用のトレースのグループを作成できます。グループを作成すると、そのグループは、トレースマップ 、トレース 、分析 の 3 ページすべてのグループドロップダウンメニューでフィルターとして使用可能になります。



グループは名前または Amazon リソースネーム (ARN) で識別され、フィルタ式を含みます。サービスは着信トレースを式と比較し、それに応じてそれらを保管します。フィルター式の作成方法の詳細については、フィルター式を使用する を参照してください。

グループのフィルタ式を更新しても、すでに記録されているデータは変わりません。更新は後続のトレースにのみ適用されます。これにより、新しい式と古い式がマージされたグラフが表示される場合があります。これを回避するには、現在のグループを削除し、新しいグループを作成します。

> **Note**
>
> グループは、フィルタ式と一致する取得済みのトレースの数で請求されます。詳細については、「AWS X-Ray 料金表」を参照してください。

グループを作成する

> **Note**
>
> Amazon CloudWatch コンソール内から X-Ray グループを設定できるようになりました。X-Ray コンソールを引き続き使用することもできます。

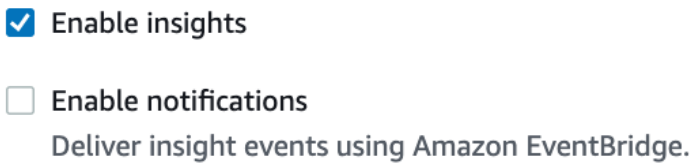
CloudWatch console

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開きます。

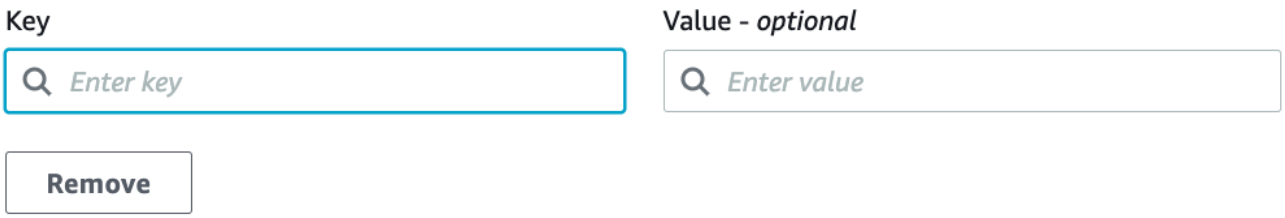
2. 左側のナビゲーションペインの [Settings] (設定) を選択します。
3. [X-Ray トレース] セクションの [グループ] の下にある [設定の表示] を選択します。
4. グループのリストの上にある [グループを作成] を選択します。
5. [Create group] (グループの作成) ページで、グループの名前を入力します。グループ名は最大 32 文字で、英数字とダッシュを使用できます。グループ名では大文字と小文字が区別されます。
6. フィルター式を入力します。フィルター式の作成方法の詳細については、フィルター式を使用する を参照してください。次の例では、グループによってサービス`api.example.com`からの障害トレースと応答時間が 5 秒以上であったサービスへのリクエストがフィルタリングされます。

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

7. [Insights] (インサイト)で、グループのインサイトアクセスを有効または無効にします。インサイトの詳細については、「X-Ray Insights を使用する」を参照してください。



8. [タグ] で [新しいタグを追加] を選択してタグキーを入力し、オプションでタグ値を入力します。必要に応じて、続けてタグを追加します。タグ キーは一意である必要があります。タグを削除するには、タグの下にある [削除] を選択します。タグの詳細については、X-Ray のサンプリングルールとグループのタグ付けを参照してください。



9. [グループを作成] を選択します。

X-Ray console

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/xray/home で X-Ray コンソールを開きます。

2. 左側のナビゲーションペインのグループページ、またはトレースマップ 、トレース 、分析のいずれかのページのグループメニューから、グループの作成ページを開きます。
3. [Create group] (グループの作成) ページで、グループの名前を入力します。グループ名は最大 32 文字で、英数字とダッシュを使用できます。グループ名では大文字と小文字が区別されます。
4. フィルター式を入力します。フィルター式の作成方法の詳細については、フィルター式を使用する を参照してください。次の例では、グループによってサービスapi.example.comからの障害トレースと応答時間が 5 秒以上であったサービスへのリクエストがフィルタリングされます。

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

5. [Insights] (インサイト)で、グループのインサイトアクセスを有効または無効にします。インサイトの詳細については、「X-Ray Insights を使用する」を参照してください。

6. [Tags] (タグ) で、タグ キー、および必要に応じてタグ値を入力します。タグを追加すると、別のタグを入力するための新しい行が表示されます。タグ キーは一意である必要があります。タグを削除するには、タグの行の最後にあるX を選択します。タグの詳細については、X-Ray のサンプリングルールとグループのタグ付けを参照してください。

7. [グループを作成] を選択します。

グループを適用します

CloudWatch console

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開きます。
2. X-Ray トレースの下のナビゲーションペインで、次のいずれかのページを開きます:

- トレースマップ
- トレース

3. [X-Ray グループでフィルタリング] フィルターにグループ名を入力します。ページに表示されるデータは、グループに設定されているフィルター式と一致するように変更されます。

X-Ray console

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/xray/home で X-Ray コンソールを開きます。
2. ナビゲーションペイン から次のいずれかのページを開きます:
   - トレースマップ
   - トレース
   - 分析
3. グループ メニューで、the section called “グループを作成する”で作成したグループを選択します。ページに表示されるデータは、グループに設定されているフィルター式と一致するように変更されます。

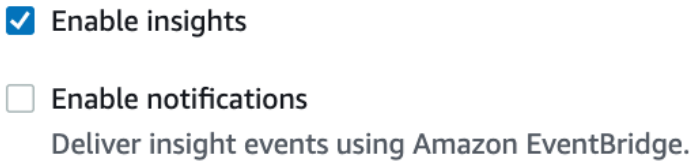
グループを編集します

CloudWatch console

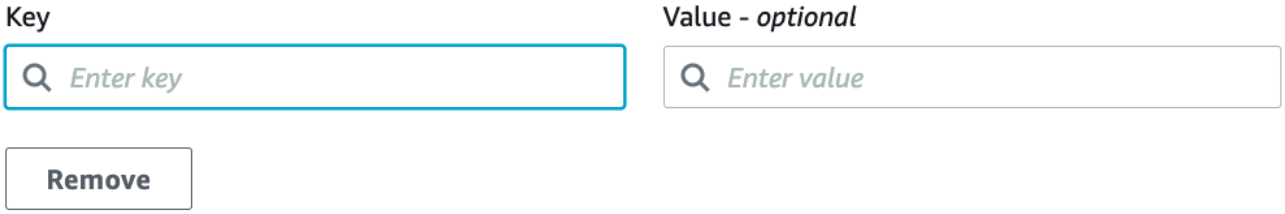
1. にサインイン AWS Management Console し、https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開きます。
2. 左側のナビゲーションペインの [Settings] (設定) を選択します。
3. [X-Ray トレース] セクションの [グループ] の下にある [設定を表示] を選択します。
4. [グループ] セクションからグループを選択し、[編集] を選択します。
5. グループの名前を変更することはできませんが、フィルター式を更新することはできます。フィルター式の作成方法の詳細については、フィルター式を使用する を参照してください。次の例では、リクエストのURLアドレスにexample/game が含まれ、リクエストの応答時間が5秒以上であるサービス api.example.com からの障害トレース を、グループでフィルタリングしています。

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

6. [Insights] (インサイト)で、グループのインサイトアクセスを有効または無効にします。インサイトの詳細については、「X-Ray Insights を使用する」を参照してください。



7. [タグ] で [新しいタグを追加] を選択してタグキーを入力し、オプションでタグ値を入力します。必要に応じて、続けてタグを追加します。タグ キーは一意である必要があります。タグを削除するには、タグの下にある [削除] を選択します。タグの詳細については、X-Ray のサンプリングルールとグループのタグ付けを参照してください。



8. グループの更新が完了したら、[グループの更新] を選択します。

X-Ray console

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/xray/home で X-Ray コンソールを開きます。
2. 次のいずれかを実行して [Edit group] (グループの編集ページ) を開きます。
   a. [Groups] (グループ) ページで、グループ名を選択して編集します。
   b. 次のいずれかのページの グループ メニューで、グループを挙げて、[Edit] (編集) を選びます。
      - トレースマップ
      - トレース
      - 分析
3. グループの名前を変更することはできませんが、フィルター式を更新することはできます。フィルター式の作成方法の詳細については、フィルター式を使用する を参照してください。次の例では、リクエストのURLアドレスに`example/game` が含まれ、リクエストの応答時間

が5秒以上であるサービス api.example.com からの障害トレース を、グループでフィルタリングしています。

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

4. Insights (インサイト)では、グループのインサイトおよびインサイト通知を有効または無効にします。インサイトの詳細については、「X-Ray Insights を使用する」を参照してください。

5. [Tags] (タグ) で、タグキーと値を編集します。タグキーは一意である必要があります。タグ値は任意であり、 必要であれば値 を削除できます。タグを削除するには、タグの行の最後にあるXを選択します。タグの詳細については、X-Ray のサンプリングルールとグループのタグ付けを参照してください。

6. グループの更新が完了したら、[グループの更新] を選択します。

グループのクローンを作成します

グループを複製すると、既存のグループのフィルター式とタグを持つ新しいグループが作成されます。グループのクローンを作成すると、新しいグループの名前はクローンの元のグループの名前に-clone 付加された名前になります。

CloudWatch console

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開きます。
2. 左側のナビゲーションペインの [Settings] (設定) を選択します。
3. [X-Ray トレース] セクションの [グループ] の下にある [設定を表示] を選択します。
4. [グループ] セクションからグループを選択し、[クローン] を選択します。

5. [Create group] (グループを作成する)ページで、グループの名前は*[group-name]* (グループ名) `-clone`。必要に応じて、グループの新しい名前を入力します。グループ名は最大 32 文字で、英数字とダッシュを使用できます。グループ名では大文字と小文字が区別されます。
6. フィルター式を既存のグループから保持するか、オプションで新しいフィルター式を入力できます。フィルター式の作成方法の詳細については、フィルター式を使用する を参照してください。次の例では、グループによってサービス `api.example.com` からの障害トレースと応答時間が 5 秒以上であったサービスへのリクエストがフィルタリングされます。

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

7. [Tags] (タグ) で、必要に応じてタグのキーと値を編集します。タグキーは一意である必要があります。タグ値は任意であり、必要に応じて値を削除できます。タグを削除するには、タグの行の最後にある[X] を選択します。タグの詳細については、X-Ray のサンプリングルールとグループのタグ付けを参照してください。
8. [グループを作成] を選択します。

X-Ray console

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/xray/home で X-Ray コンソールを開きます。
2. 左側のナビゲーションペインから [グループ] ページを開き、クローンを作成するグループの名前を選択します。
3. [アクション] メニューから、[グループのクローン作成] を選択します。
4. [Create group] (グループを作成する)ページで、グループの名前は*[group-name]* (グループ名) `-clone`。必要に応じて、グループの新しい名前を入力します。グループ名は最大 32 文字で、英数字とダッシュを使用できます。グループ名では大文字と小文字が区別されます。
5. フィルター式を既存のグループから保持するか、オプションで新しいフィルター式を入力できます。フィルター式の作成方法の詳細については、フィルター式を使用する を参照してください。次の例では、グループによってサービス `api.example.com` からの障害トレースと応答時間が 5 秒以上であったサービスへのリクエストがフィルタリングされます。

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

6. [Tags] (タグ) で、必要に応じてタグのキーと値を編集します。タグキーは一意である必要があります。タグ値は任意であり、必要に応じて値を削除できます。タグを削除するには、タ

グの行の最後にある[X] を選択します。タグの詳細については、X-Ray のサンプリングルールとグループのタグ付けを参照してください。

7. [グループを作成] を選択します。

グループの削除

グループを削除するには、このセクションの手順に従います。[Default] (デフォルト) グループを削除することはできません。

CloudWatch console

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開きます。
2. 左側のナビゲーションペインの [Settings] (設定) を選択します。
3. [X-Ray トレース] セクションの [グループ] の下にある [設定を表示] を選択します。
4. [グループ] セクションからグループを選択し、[削除] を選択します。
5. 確認を求められたら [Delete] を選択します。

X-Ray console

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/xray/home で X-Ray コンソールを開きます。
2. 左側のナビゲーションペインから [グループ] ページを開き、削除するグループの名前を選択します。
3. [Actions] メニューで、[Delete] を選択します。
4. 確認を求められたら [Delete] を選択します。

Amazon でグループメトリクスを表示する CloudWatch

グループの作成後、着信トレースは、X-Ray サービスに格納されるときにグループのフィルター式と照合されます。各条件に一致するトレース数のメトリクスは、 CloudWatch 毎分 Amazon に発行されます。グループの編集ページでメトリクスの表示を選択すると、 CloudWatchコンソールがメトリクスページに開きます。 CloudWatch メトリクスの使用方法の詳細については、「Amazon CloudWatch ユーザーガイド」の「Amazon メトリクスの使用 CloudWatch 」を参照してください。

CloudWatch console

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開きます。
2. 左側のナビゲーションペインの [Settings] (設定) を選択します。
3. [X-Ray トレース] セクションの [グループ] の下にある [設定を表示] を選択します。
4. [グループ] セクションからグループを選択し、[編集] を選択します。
5. [Edit group] (グループ編集) ページで、[View metric] (メトリック表示) を選択します。

   CloudWatch コンソールのメトリクスページが新しいタブで開きます。

X-Ray console

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/xray/home で X-Ray コンソールを開きます。
2. 左側のナビゲーションペインから [グループ] ページを開き、メトリクスを表示するグループの名前を選択します。
3. [Edit group] (グループ編集) ページで、[View metric] (メトリック表示) を選択します。

   CloudWatch コンソールのメトリクスページが新しいタブで開きます。

サンプリングルールを設定する

AWS X-Ray コンソールを使用して、サービスのサンプリングルールを設定できます。サンプリング設定によるアクティブトレース AWS のサービス をサポートする X-Ray SDK と は、サンプリングルールを使用して、記録するリクエストを決定します。

サンプリングルールを設定する

以下のユースケースのサンプリングを設定できます。

- API Gateway Entrypoint — API Gateway は、サンプリングとアクティブトレースをサポートします。API ステージでアクティブトレースを有効にする方法については、「の Amazon API Gateway アクティブトレースのサポート AWS X-Ray」を参照してください。
- AWS AppSync – サンプリングとアクティブトレース AWS AppSync をサポートします。 AWS AppSync リクエストでアクティブトレースを有効にするには、「X-Ray による AWS トレース」を参照してください。

- コンピューティングプラットフォームで X-Ray SDK を計測する – Amazon EC2、Amazon ECS、などのコンピューティングプラットフォームを使用する場合 AWS Elastic Beanstalk、アプリケーションが最新の X-Ray SDK で計測されたときにサンプリングがサポートされます。

サンプリングルールのカスタマイズ

サンプリングルールをカスタマイズすることで、記録するデータの量を制御できます。また、コードを変更したり再デプロイしたりすることなく、サンプリング動作を変更することもできます。　サンプリングルールにより、X-Ray SDK に一連の基準に対して記録するリクエスト数を指示します。デフォルトでは、X-Ray SDK は 1 秒ごとに最初に受信したリクエストと、追加のリクエストの 5% を記録します。1 秒あたり 1 つのリクエストがリザーバです。これにより、サービスがリクエストを処理している限り、毎秒少なくとも 1 つのトレースが記録されます。5% は、リザーバサイズを超えて追加リクエストがサンプリングされるレートです。

X-Ray SDK を設定して、使用するコードに含める JSON ドキュメントからサンプリングルールを読み取ることができます。ただし、サービスで複数のインスタンスを実行するときに、各インスタンスは個別にサンプリングを実行します。これによりサンプリングされたリクエストの全体的な割合 (パーセント) が増加します。すべてのインスタンスのリザーバが実質的に結合されるからです。さらに、ローカルサンプリングルールを更新するために、コードを再デプロイする必要があります。

X-Ray コンソールでサンプリングルールを定義し、[configuring the SDK] (SDK を設定)して、X-Ray サービスからルールを読み取ることで、これら両方の問題を回避できます。このサービスでは、各ルールのリザーバを管理し、実行されているインスタンスの数に基づいて、リザーバを均等に分散させるため、使用するサービスの各インスタンスにクォータを割り当てます。リザーバの制限は、設定したルールに従って計算されます。サービスでルールが設定されているので、追加のデプロイを行わずにルールを管理できます。 AWS SDK の詳細については、「 SDK を使用する」を参照してください。

> **Note**
>
> X-Ray では、ベストエフォート型の方法でサンプリングルールを適用しているため、場合によっては、有効なサンプリングレートが、設定されたサンプリングルールと完全に一致しないことがあります。しかし、時間の経過とともに、サンプリングされるリクエスト数が設定したパーセンテージに近づくはずです。

Amazon CloudWatch コンソール内から X-Ray サンプリングルールを設定できるようになりました。X-Ray コンソールを引き続き使用することもできます。

CloudWatch console

CloudWatch コンソールでサンプリングルールを設定するには

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開きます。
2. 左側のナビゲーションペインの [Settings] (設定) を選択します。
3. [X-Ray トレース] セクションの [サンプリングルール] の下にある [設定を表示] を選択します。
4. ルールを作成するには、[サンプリングルールの作成] を選択します。

   ルールを編集するには、ルールを選択してから [編集] を選択します。

   ルールを削除するには、ルールを選択してから [削除] を選択します。

X-Ray console

X-Ray コンソールでのサンプリングルールを設定するには

1. [X-Ray console] (X-Ray コンソール)を開きます。
2. 左側のナビゲーションペインで [サンプリング] を選択します。
3. ルールを作成するには、[Create sampling rule] (サンプリングルールの作成) を選択します。

   ルールを編集するには、ルールの名前を選択します。

   ルールを削除するには、ルールを選択し、[Actions] (アクション) メニューを使用して削除します。

サンプリングルールオプション

次のオプションが各ルールで利用できます。文字列値では、ワイルドカードを使用して、1 つの文字列 (?) またはゼロ以上の文字 (*) に一致させることができます。

サンプリングルールオプション

- [Rule name] (ルール名) (文字列) - ルールの一意の名前。
- [Priority] (優先度) (1 ~ 9999 の整数) - サンプリングルールの優先度。サービスでは、優先度の昇順でルールが評価され、一致する最初のルールを使用してサンプリングの決定が行われます。

- [Reservoir] (リザーバ) (負ではない整数) - 固定レートを適用する前に、1 秒あたりに計測する、一致するリクエストの固定数。リザーバはサービスで直接使用されませんが、ルールを一括して使用するすべてのサービスに適用されます。
- [レート] (0 ~ 100) - リザーバの上限に達した後に機器と一致するリクエストのパーセンテージ。コンソールでサンプリングルールを設定するときは、0 から 100 までのパーセンテージを選択します。JSON ドキュメントを使用してクライアント SDK でサンプリングルールを設定する場合は、0 から 1 までのパーセンテージ値を指定します。
- サービス名 (文字列) — トレースマップに表示される、計測されたサービスの名前。
  - X-Ray SDK - レコーダーで設定したサービス名。
  - Amazon API Gateway - *`api-name`*/*`stage`*。
- サービスタイプ (文字列) — トレースマップに表示されるサービスタイプ。X-Ray SDK の場合、適切なプラグインを適用することで、サービスタイプを設定します。
  - `AWS::ElasticBeanstalk::Environment` – AWS Elastic Beanstalk 環境 (プラグイン) 。
  - `AWS::EC2::Instance` — Amazon EC2 インスタンス (プラグイン)。
  - `AWS::ECS::Container` — Amazon ECS コンテナ (プラグイン)。
  - `AWS::APIGateway::Stage` - Amazon API Gateway ステージ。
  - `AWS::AppSync::GraphQLAPI` – AWS AppSync API リクエスト。
- [Host] (ホスト) (文字列) - HTTP ホスト ヘッダーからの ホスト名。
- [HTTP method] (HTTP メソッド) (文字列) - HTTP リクエストのメソッド。
- [URL path] (URL パス) (文字列) - リクエストの URL パス。
  - X-Ray SDK - HTTP リクエスト URL のパス部分。
- リソース ARN (文字列) — サービスを実行している AWS リソースの ARN。
  - X-Ray SDK — サポートされていません。SDK では [リソース ARN] に * が設定されているルールのみを使用できます。
  - Amazon API Gateway — ステージ ARN。
- (オプション) [Attributes] (属性) (キーと値) - サンプリングデシジョンが行われたときに認識されるセグメント属性。
  - X-Ray SDK — サポートされていません。SDK は属性を指定するルールを無視します。
  - Amazon API Gateway — 元の HTTP リクエストからのヘッダー。

サンプリングルールの例

Example - リザーバなし、低レートのデフォルトルール

デフォルトルールのリザーバとレートを変更することができます。他のルールに一致しないリクエストにデフォルトのルールが適用されます。

- [Reservoir] (リザーバ): **0**
- レート: **5** (**0.05** JSON ドキュメントを使用して設定した場合)

Example - 問題のあるルートのすべてのリクエストをトレースするデバッグルール

デバッグ用に一時的に適用される、高優先度のルールです。

- [Rule name] (ルール名): **DEBUG – history updates**
- [Priority] (優先度): **1**
- [Reservoir] (リザーバ): **1**
- レート: **100** (**1** JSON ドキュメントを使用して設定した場合)
- [Service name] (サービス名): **Scorekeep**
- [Service type] (サービスタイプ): *****
- [Host] (ホスト): *****
- [HTTP method] (HTTP メソッド): **PUT**
- [URL path] (URL パス): **/history/***
- [Resource ARN] (リソース ARN): *****

Example - POST 用の最小レートが高い

- [Rule name] (ルール名): **POST minimum**
- [Priority] (優先度): **100**
- [Reservoir] (リザーバ): **10**
- レート: **10** (**.1** JSON ドキュメントを使用して設定した場合)
- [Service name] (サービス名): *****
- [Service type] (サービスタイプ): *****
- [Host] (ホスト): *****

- [HTTP method] (HTTP メソッド): **POST**
- [URL path] (URL パス): *****
- [Resource ARN] (リソース ARN): *****

サンプリングルールを使用するようにサービスを設定する

X-Ray SDK ではコンソールで設定したサンプリングルールを使用するため追加の設定が必要です。サンプリング戦略を設定する方法の詳細については、使用言語の「設定」トピックを参照してください。

- Java: サンプリングルール
- Go: サンプリングルール
- Node.js: サンプリングルール
- Python: サンプリングルール
- Ruby: サンプリングルール
- .NET: サンプリングルール

API Gatewayについては、の Amazon API Gateway アクティブトレースのサポート AWS X-Ray を参照してください。

サンプリング結果の表示

X-Ray コンソールの [Sampling] (サンプリング) ページには、サービスでの各サンプリングルールの使用方法に関する詳細情報が表示されます。

[Trend (トレンド)] 列には、直近の数分でルールがどのように使用されたのかを表示します。各列に、10 秒ウィンドウの統計が表示されます。

サンプリング統計

- [Total matched rule] (ルールに一致した総数): 対象ルールに一致するリクエストの数。この数には、対象ルールに一致する可能性があるすべてのリクエストが含まれるわけではありません。優先度の高いルールに最初に一致したリクエストのみが含まれます。
- [Total sampled] (サンプリングされた総数): 記録されたリクエスト数。
- [Sampled with fixed rate] (サンプリングの固定レート): ルールの固定レートを適用してサンプリングされたリクエスト数。

- [Sampled with reservoir limit] (サンプリングのリザーバ制限):X-Rayによって割り当てられたクォータを使用してサンプリングされたリクエスト数。
- [Borrowed from reservoir] (リザーバから借用): リザーバからの借用によって、サンプリングされたリクエスト数。サービスで初めてリクエストがルールに一致するとき、X-Rayによりクォータがまだ割り当てられていません。ただし、リザーバが少なくとも 1 である場合、X-Rayがクォータを割り当てるまで、サービスは 1 秒あたり 1 つのトレースを借用します。

サンプリングの統計およびサービスがサンプリングルールを使用する方法の詳細については、「X-Ray API でのサンプリングルールの使用」を参照してください。

次のステップ

X-Ray API を使用して、サンプリングルールを管理できます。API では、スケジュールに従って、またはアラームや通知に応答して、プログラムでルールを作成および更新することができます。手順と追加のルールの例については、X-Ray API を使用したサンプリング、グループ、および暗号化設定の構成 を参照してください。

X-Ray SDK および は、X-Ray API を使用して、サンプリングルールの読み取り、サンプリング結果のレポート、サンプリングターゲットの取得 AWS のサービス を行います。X-Ray がサービスにクォータを割り当てていないルールにリクエストが一致するとき、サービスは、各ルールの適用、優先度に基づくルールの評価、およびリザーバからの借用に関する頻度を追跡する必要があります。サービスが API をサンプリングに使用する方法の詳細については、「」を参照してくださいX-Ray API でのサンプリングルールの使用。

X-Ray SDK がサンプリング API を呼び出すと、プロキシとして X-Ray デーモンを使用します。TCP ポート 2000 をすでに使用している場合、別のポートでプロキシを実行するようにデーモンを設定できます。詳細については、「AWS X-Ray デーモンの設定」を参照してください。

コンソールのディープリンク

ルートとクエリを使用して、特定のトレース、またはトレースとトレースマップのフィルタリングされたビューにディープリンクできます。

コンソールページ

- ようこそページ - [xray/home#/welcom] (xray/ホーム#/ようこそ)
- はじめに - [xray/home#/getting-starte] (xray/ホーム#/はじめに)
- トレースマップ – xray/home#/service-map
- トレース - [xray/home#/trace] (xray/ホーム#/トレース)

トレース

個別のトレースのタイムライン、raw、マップビューのリンクを生成できます。

[Trace timeline] (トレースのタイムライン) - `xray/home#/traces/`*`trace-id`*

[Raw trace data] (未加工のトレースデータ) - `xray/home#/traces/`*`trace-id`*`/raw`

Example - 未加工のトレースデータ

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

フィルタ式

フィルタリングされたトレースリストへのリンク

[Filtered traces view] (フィルタリングされたトレースビュー) - `xray/home#/traces?filter=`*`filter-expression`*

Example - フィルター表現

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

Example - フィルター表現 (URL エンコード)

```
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
```

フィルタ式の詳細については、「フィルター式を使用する」を参照してください。

[Time range] (時間範囲)

期間または開始時刻と終了時刻を ISO8601 形式で指定します。時間範囲は UTC で、最大で6時間にすることができます。

[Length of time] (期間) - `xray/home#/`*`page`*`?timeRange=`*`range-in-minutes`*

Example – 過去 1 時間のトレースマップ

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

[Start and end time] (開始と終了の時刻) - `xray/home#/page?timeRange=start~end`

Example - 秒単位の正確な時間範囲

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

Example - 分単位の正確な時間範囲

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

リージョン

を指定 AWS リージョン して、そのリージョンのページにリンクします。リージョンを指定しない場合、コンソールは最後に利用したリージョンにリダイレクトされます。

[Region] (リージョン) - `xray/home?region=region#/page`

Example – 米国西部 (オレゴン) (us-west-2) のトレースマップ

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

その他のクエリパラメーターを使用してリージョンを含めると、リージョンクエリは、ハッシュ前、X-Ray 固有のクエリはページ名の後に指定されます。

Example – 米国西部 (オレゴン) (us-west-2) の過去 1 時間のトレースマップ

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

結合

Example - 期間フィルターを含む最近のトレース

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

出力

- ページ - トレース

- 時間範囲 - 最後の 15 分
- フィルター - 期間 >= 5 および 期間 <= 8

SDK を使用する

コマンドラインインターフェイスを使用する場合、または で使用可能なものよりも多くのカスタムトレース、モニタリング、またはログ記録機能が必要な場合は、 SDK を使用します AWS Management Console。 AWS SDK を使用して、X-Ray APIs を使用するプログラムを開発することもできます。Distro for OpenTelemetry (ADOT) SDK AWS または X-Ray SDK を使用できます。

SDK を使用する場合は、アプリケーションの計測時とコレクターまたはエージェントの設定時の両方で、ワークフローにカスタマイズを追加できます。SDK を使用して、 ではできない以下のタスクを実行できます AWS Management Console。

- カスタムメトリクスの発行 – 1 秒までの高解像度のサンプルメトリクス、複数のディメンションを使用してメトリクスに関する情報を追加し、データポイントを統計セットに集約します。
- コレクターのカスタマイズ — レシーバー、プロセッサ、エクスポーター、コネクタなど、コレクターの任意の部分の設定をカスタマイズします。
- 計測をカスタマイズする – セグメントとサブセグメントをカスタマイズし、カスタムキーと値のペアを属性として追加し、カスタムメトリクスを作成します。
- サンプリングルールをプログラムで作成および更新します。

AWS セキュリティと最適化のレイヤーを追加した標準化された ADOT SDK を柔軟に使用する場合は、 OpenTelemetry SDK を使用します。 AWS Distro for OpenTelemetry （ ADOT) SDK はベンダーに依存しないパッケージで、コードを再計測しなくても、他のベンダーやAWS 以外のサービスのバックエンドと統合できます。

X-Ray SDK を既に使用していて、バックエンドと AWS のみ統合し、X-Ray またはアプリケーションコードの操作方法を変更しない場合は、X-Ray SDK を使用します。

各機能の詳細については、「」を参照してくださいAWS Distro for OpenTelemetry と X-Ray SDKs の選択。

ADOT SDK を使用する

ADOT SDK は、バックエンドサービスにデータを送信する一連のオープンソース APIsライブラリ、エージェントです。 ADOTは でサポートされており AWS、複数のバックエンドとエージェントと統

合され、OpenTelemetryコミュニティによって管理される多数のオープンソースライブラリを提供します。ADOT SDK を使用してアプリケーションを計測し、ログ、メタデータ、メトリクス、トレースを収集します。ADOT を使用して サービスをモニタリングし、 のメトリクスに基づいてアラームを設定することもできます CloudWatch。

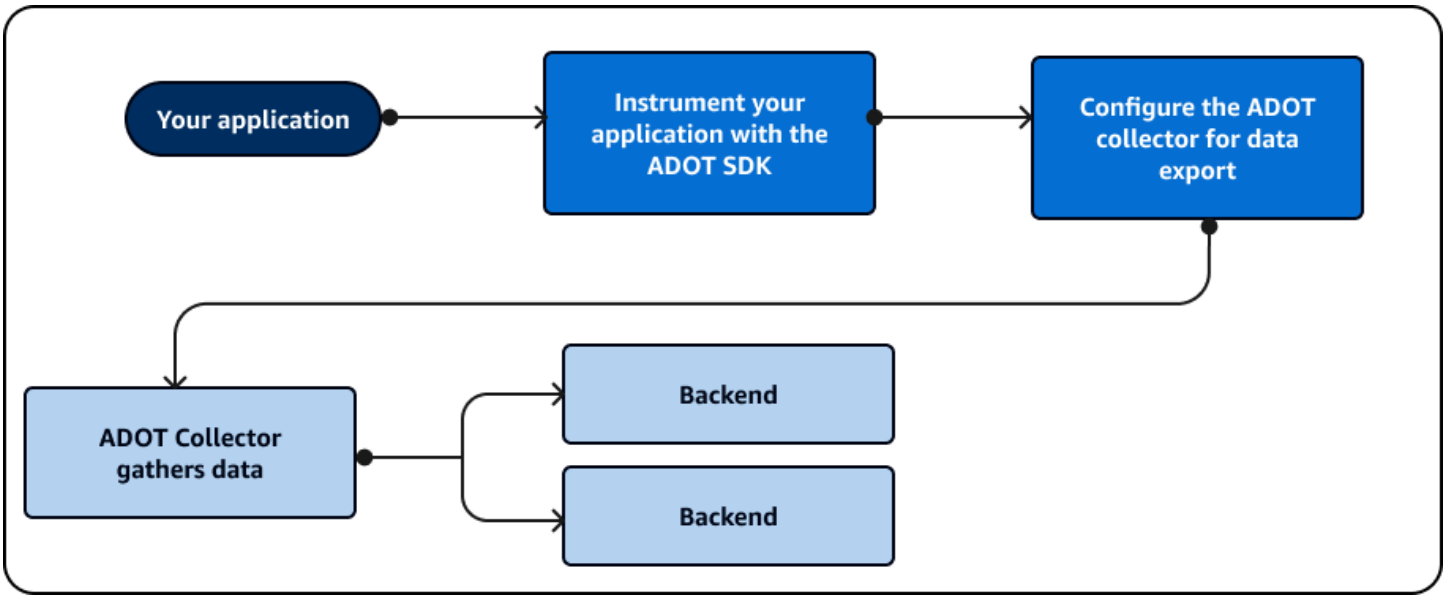
ADOT SDK を使用している場合は、 エージェントと組み合わせて次のオプションを使用できます。

- CloudWatch エージェントで ADOT SDK を使用する – 推奨。
- ADOT コレクターで ADOT SDK を使用する — セキュリティと最適化の AWS レイヤーでベンダーに依存しないソフトウェアを使用する場合は、 が推奨します。

ADOT SDK を使用するには、次の手順を実行します。

- ADOT SDK を使用してアプリケーションを計測します。詳細については、ADOT 技術ドキュメントのプログラミング言語のドキュメントを参照してください。
- ADOT コレクターを設定して、収集するデータの送信先を指定します。

ADOT コレクターは、データを受信すると、ADOT設定で指定したバックエンドに送信します。 は AWS、次の図に示すように、 以外のベンダーを含む複数のバックエンドにデータを送信ADOTできます。



AWS は定期的に を更新ADOTして機能を追加し、OpenTelemetryフレームワークと連携させます。開発のための更新と将来の計画は、一般に公開されているロードマップの一部ADOTです。 は、以下を含む複数のプログラミング言語ADOTをサポートしています。

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

Python を使用している場合、 ADOTはアプリケーションを自動的に計測できます。の使用を開始するにはADOT、「 Distro for Collector の概要」と「使用開始」を参照してください。 AWS OpenTelemetry

X-Ray SDK を使用する

X-Ray SDK は、バックエンドサービスに AWS データを送信する AWS APIsとライブラリです。X-Ray SDK を使用してアプリケーションを計測し、トレースデータを収集します。X-Ray SDK を使用して ログまたはメトリクスデータを収集することはできません。

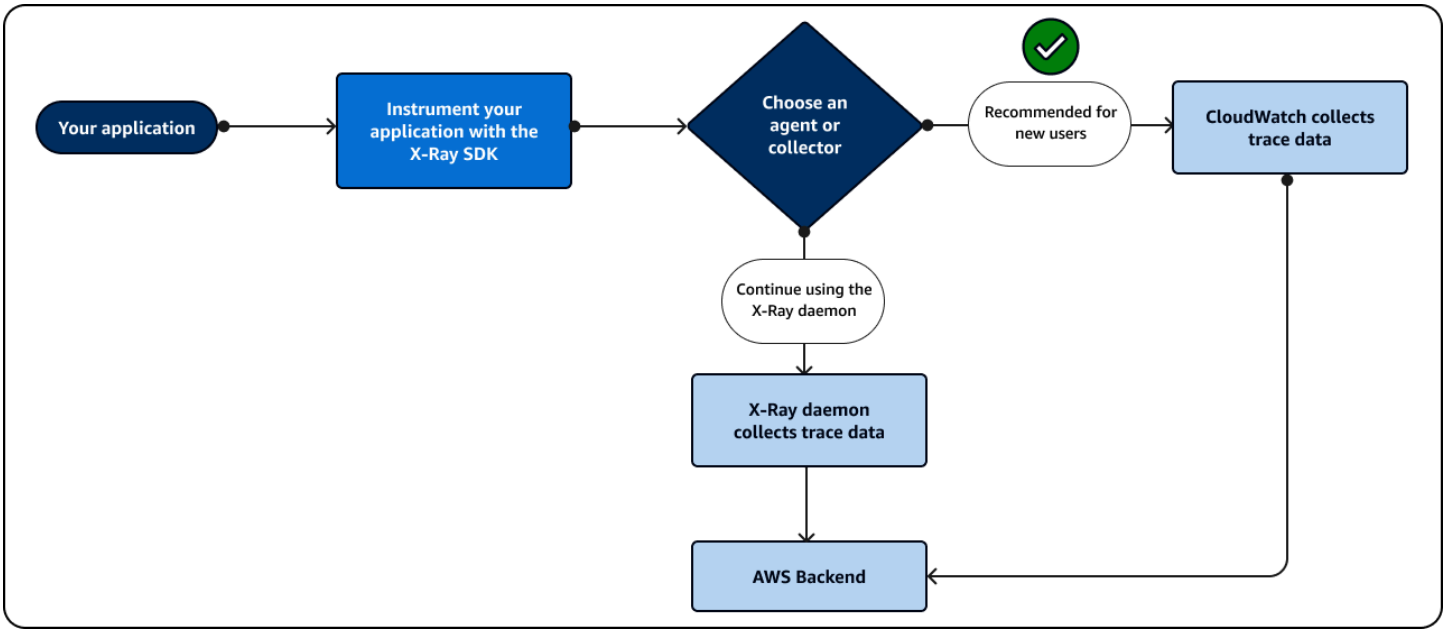
X-Ray SDK を使用している場合は、 エージェントと組み合わせて以下のオプションがあります。

- で X-Ray SDK を使用する AWS X-Ray デーモン – アプリケーションコードを更新しない場合は、これを使用します。
- CloudWatch エージェントで X-Ray SDK を使用する – (推奨) CloudWatch エージェントは X-Ray SDK と互換性があります。

X-Ray SDK を使用するには、次の手順を実行します。

- X-Ray SDK を使用してアプリケーションを計測します。
- コレクターを設定して、収集するデータの送信先を指定します。 CloudWatch エージェントまたは X-Ray デーモンを使用してトレース情報を収集できます。

コレクターまたはエージェントがデータを受信すると、エージェント設定で指定した AWS バックエンドに送信されます。X-Ray SDK は、次の図に示すように、 AWS バックエンドにのみデータを送信できます。



を使用している場合はJava、X-Ray SDK を使用してアプリケーションを自動的に計測できます。X-Ray SDK の使用を開始するには、次のプログラミング言語に関連付けられているライブラリを参照してください。

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

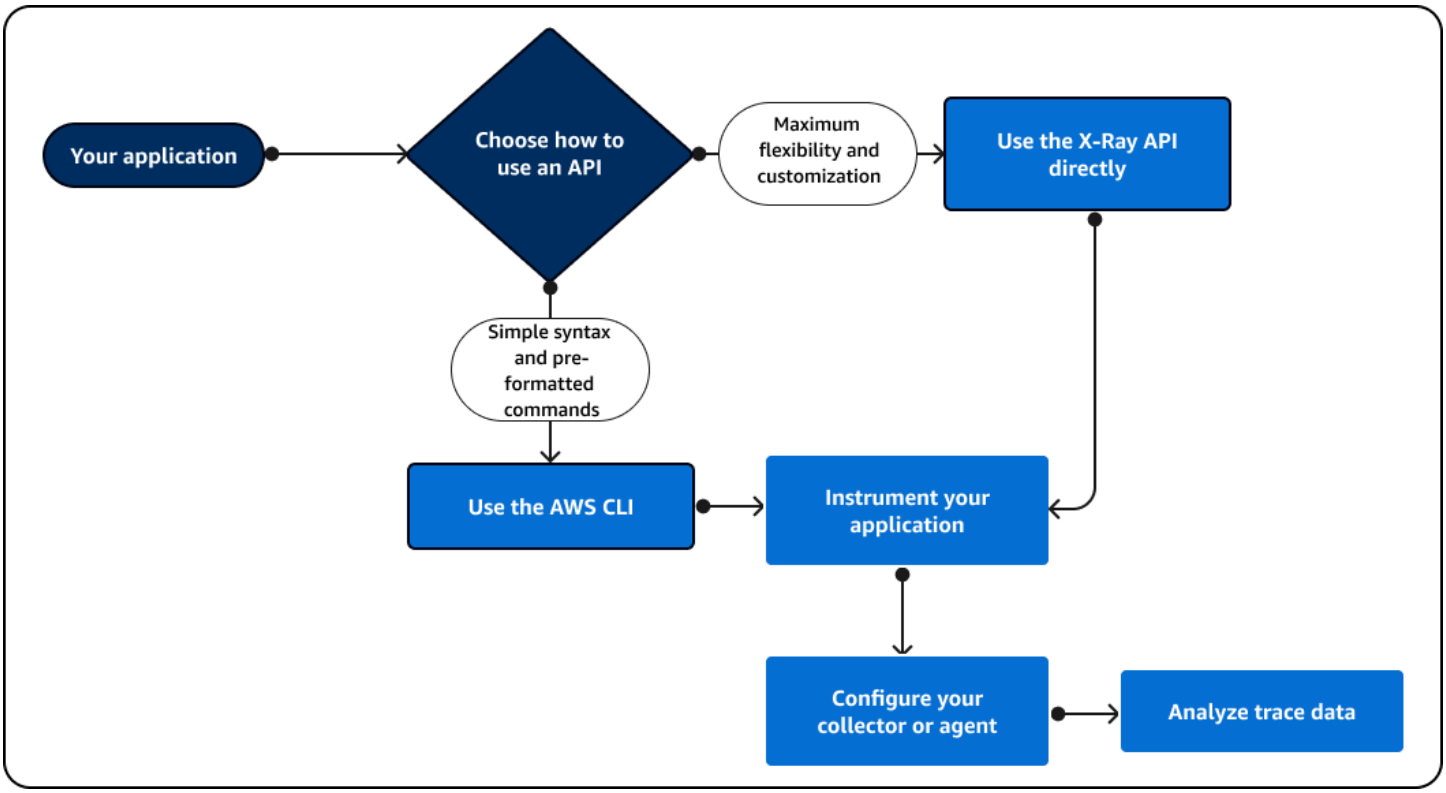
X-Ray API を使用する

X-Ray SDK がプログラミング言語をサポートしていない場合は、X-Ray APIs を直接使用するか、AWS Command Line Interface （ AWS CLI) を使用して X-Ray API コマンドを呼び出すことができます。次のガイダンスを使用して、API の操作方法を選択します。

- 事前フォーマットされたコマンドを使用するか、リクエスト内のオプションを使用して、構文 AWS CLI を簡素化するには、 を使用します。
- X-Ray API を直接使用して、X-Ray に対して行うリクエストの柔軟性とカスタマイズを最大化します。

の代わりに X-Ray API を直接使用する場合は AWS CLI、リクエストを正しいデータ形式でパラメータ化する必要があり、認証とエラー処理を設定する必要もあります。

次の図は、X-Ray API を操作する方法を選択するガイダンスを示しています。



X-Ray API を使用して、トレースデータを X-Ray に直接送信します。X-Ray API は、以下の一般的なアクションを含め、X-Ray SDK で使用できるすべての関数をサポートします。

- PutTraceSegments – セグメントドキュメントを X-Ray にアップロードします。
- BatchGetTraces – トレース IDsのリスト内のトレースのリストを取得します。取得された各トレースは、1 つのリクエストからのセグメントドキュメントのコレクションです。
- GetTraceSummaries – トレースIDs と注釈を取得します。を指定`FilterExpression`して、トレース概要のサブセットを取得できます。
- GetTraceGraph – 特定のトレース ID のサービスグラフを取得します。
- GetServiceGraph – 受信リクエストを処理し、ダウンストリームリクエストを呼び出す のサービスを説明するJSONフォーマットされたドキュメントを取得します。

アプリケーションコード内で AWS Command Line Interface （ AWS CLI) を使用して、プログラムで X-Ray とやり取りすることもできます。 AWS CLI は、X-Ray SDK で使用できるすべての関数をサ

ポートします。これには、他の 用の関数も含まれます AWS のサービス。次の関数は、前述の API オペレーションのバージョンで、形式がよりシンプルです。

- put-trace-segments – セグメントドキュメントを X-Ray にアップロードします。
- batch-get-traces – トレース IDsのリスト内のトレースのリストを取得します。取得された各トレースは、1 つのリクエストからのセグメントドキュメントのコレクションです。
- get-trace-summaries – トレースIDs と注釈を取得します。を指定`FilterExpression`して、トレース概要のサブセットを取得できます。
- get-trace-graph – 特定のトレース ID のサービスグラフを取得します。
- get-service-graph – 受信リクエストを処理し、ダウンストリームリクエストを呼び出す のサービスを説明する`JSON`フォーマットされたドキュメントを取得します。

開始するには、オペレーティングシステムAWS CLIの をインストールする必要があります。 は Linux、、 macOS オペレーティングシステム AWS をサポートしていますWindows。X-Ray コマンドのリストの詳細については、AWS CLI 「X-Ray の コマンドリファレンスガイド」を参照してください。

X-Ray API の詳細

X-Ray API は、 AWS SDK を介して AWS Command Line Interface、または HTTPS 経由で直接、すべての X-Ray 機能へのアクセスを提供します。X-Ray API リファレンスに、各 API アクションの入力パラメータ、およびフィールドと返されるデータ型が記載されています。

AWS SDK を使用して、X-Ray API を使用するプログラムを開発できます。X-Ray コンソールと X-Ray デーモンはどちらも AWS SDK を使用して X-Ray と通信します。各言語の AWS SDK には、X-Ray API アクションとタイプにマッピングするクラスとメソッドのリファレンスドキュメントがあります。

AWS SDK リファレンス

- Java – AWS SDK for Java
- JavaScript – AWS SDK for JavaScript
- .NET – AWS SDK for .NET
- Ruby – AWS SDK for Ruby
- Go – AWS SDK for Go
- PHP – AWS SDK for PHP

- Python – AWS SDK for Python (Boto)

AWS Command Line Interface は、 SDK for Python を使用して AWS APIsを呼び出すコマンドラインツールです。 AWS API を初めて学習する場合、 は利用可能なパラメータを調べ、JSON またはテキスト形式でサービス出力を表示する簡単な方法 AWS CLI を提供します。

`aws xray` サブAWS CLI コマンドの詳細については、「 コマンドリファレンス」を参照してください。

AWS CLI での X-Ray API の使用

AWS CLI を使用すると、X-Ray サービスに直接アクセスし、X-Ray コンソールがサービスグラフと raw トレースデータを取得するために使用するのと同じ APIs を使用できます。サンプルアプリケーションには、 AWS CLI でこれらの APIs を使用する方法を示すスクリプトが含まれています。

前提条件

このチュートリアルでは、Scorekeep サンプルアプリケーションと、それに含まれるトレーシングデータとサービスマップを生成するスクリプトを使用します。サンプルアプリケーションチュートリアルの指示に従って、アプリケーションを起動します。

このチュートリアルでは、 AWS CLI を使用して X-Ray API の基本的な使用法を示します。Windows、Linux、および OS-X で使用できる AWS CLI は、すべての のパブリック APIs へのコマンドラインアクセスを提供します AWS のサービス。

> **Note**
>
> AWS CLI が、サンプルアプリケーションが作成されたリージョンと同じリージョンに設定されていることを確認する必要があります。

サンプルアプリケーションに含まれるテスト用のスクリプトは、cURL を使用してトラフィックを API および `jq` に送信し、出力を解析します。`jq` 実行可能ファイルは stedolan.github.io から、`curl` 実行可能ファイルは https://curl.haxx.se/download.html からダウンロードできます。ほとんどの Linux および OS X インストールには cURL が含まれています。

トレースデータの生成

ゲームの進行中、ウェブアプリケーションは数秒ごとに API にトラフィックを生成し続けますが、生成されるリクエストのタイプは 1 つだけです。test-api.sh スクリプトを使用して、エンドツーエンドのシナリオを実行し、API のテスト中により多様なトレースデータを生成します。

test-api.sh スクリプトを使用するには

1. Elastic Beanstalk コンソールを開きます。
2. 環境に対応するマネジメントコンソールに移動します。
3. 環境 [URL] をページヘッダーからコピーします。
4. bin/test-api.sh を開いて API の値を環境の URL に置き換えます。

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

5. スクリプトを実行して API へのトラフィックを生成します。

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

X-Ray API を使用する

AWS CLI は、GetServiceGraphや など、X-Ray が提供するすべての API アクションのコマンドを提供しますGetTraceSummaries。サポートされているすべてのアクションおよびそこで使用するデータ型の詳細については、「AWS X-Ray API リファレンス」を参照してください。

Example bin/service-graph.sh

```
EPOCH=$(date +%s)
```

```
aws xray get-service-graph --start-time $(($EPOCH-600)) --end-time $EPOCH
```

このスクリプトは直近 10 分のサービスグラフを取得します。

```
~/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'
```

このスクリプトは 1 分前から 2 分前の間に生成されたトレースの URL を取得します。

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

Example bin/full-traces.sh

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

このスクリプトは 1 分前から 2 分前の間に生成されたトレース全体を取得します。

```
~/eb-java-scorekeep$ ./bin/full-traces.sh | less
[
    {
        "Segments": [
            {
                "Id": "3f212bc237bafd5d",
                "Document": "{\"id\":\"3f212bc237bafd5d\",\"name\":\"DynamoDB\",
\"trace_id\":\"1-5828d9f2-a90669393f4343211bc1cf75\",\"start_time\":1.479072242459E9,
```

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


クリーンアップ

Elastic Beanstalk 環境を終了し、Amazon EC2 インスタンス、DynamoDB テーブル、およびその他のリソースをシャットダウンします。

Elastic Beanstalk 環境を終了するには

1. Elastic Beanstalk コンソールを開きます。
2. 環境に対応するマネジメントコンソールに移動します。
3. [アクション] を選択します。
4. [Terminate Environment] を選択します。
5. [Terminate] (終了) を選択します。

30日 後、トレースデータは自動的に X-Ray から削除されます。

トレースデータを X-Ray に送信する

トレースデータは、セグメントドキュメントの形式で X-Ray に送信できます。セグメントドキュメントは、アプリケーションがリクエストのサービスで行う作業に関する情報を含む JSON 形式の文

字列です。セグメント内で行われる作業、またはサブセグメントのダウンストリームサービスおよびリソースを使用する作業に関するデータはアプリケーションに記録できます。

セグメントは、アプリケーションで行われる作業に関する情報を記録します。セグメントには、少なくとも、タスク、名前、2 つの ID で使用される時間が記録されます。トレース ID は、サービス間でやり取りされるリクエストを追跡します。セグメント ID は、単一のサービスのリクエストで行われる作業を追跡します。

Example 最小完了セグメント

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

リクエストを受信したら完了するまで、プレースホルダーとして進行中のセグメントを送信できます。

Example 進行中セグメント

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

セグメントは、「PutTraceSegments」、または「X-Ray デーモンを通じて」直接 X-Ray に送信できます。

ほとんどのアプリケーションは、 AWS SDK を使用して他の サービスを呼び出すか、リソースにアクセスします。サブセグメントのダウンストリーム呼び出しに関する情報を記録します。X-Ray はサブセグメントを使用して、セグメントを送信しないダウンストリームサービスを識別し、そのエントリをサービスグラフに作成します。

サブセグメントはフルセグメントドキュメントに埋め込むことも、個別に送信することもできます。サブセグメントを個別に送信して、長期実行されているリクエストのダウンストリーム呼び出しを非

同期でトレースしたり、セグメントドキュメントの最大サイズ (64 kB) を超えないようにしたりできます。

Example サブセグメント

サブセグメントには subsegment の type および親セグメントを識別する parent_id があります。

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

セグメントとサブセグメントに含めることができるフィールドと値の詳細については、「X-Ray セグメントドキュメント」を参照してください。

トレース ID を生成する

X-Ray にデータを送信するには、リクエストごとに一意のトレース ID を生成する必要があります。

X-Ray トレース ID 形式

X-Ray trace_id は、ハイフンで区切られた 3 つの数字で構成されています。例えば、1-58406520-a006649127e371903a2de979 と指定します。これには、以下のものが含まれます：

- バージョン番号。 1
- 8 桁の 16 進数を使用した Unix エポック時間での元のリクエストの時刻。

  例えば、2016 年 12 月 1 日午前 10:00 PST のエポックタイムは1480615200秒、16 58406520 進数です。
- 24 桁の 16 進数のトレースのグローバルに一意の 96 ビット識別子。

> **Note**
>
> X-Ray は、 OpenTelemetry および W3C トレースIDs をサポートするようになりました。 W3C W3C トレース ID は、X-Ray に送信するときに X-Ray トレース ID 形式でフォーマットする必要があります。例えば、W3C トレース ID は、X-Ray に送信する`1-4efaaf4d-1e8720b39541901950019ee5`ときに としてフォーマット`4efaaf4d1e8720b39541901950019ee5`する必要があります。X-Ray トレース IDs には、Unix エポックタイムの元のリクエストタイムスタンプが含まれますが、これは W3C トレース IDs X-Ray 形式で送信する場合には必要ありません。

テスト用の X-Ray トレース ID を生成するためのスクリプトを記述することができます。これらはその 2 つの例です。

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"
```

トレース ID を作成し、X-Ray デーモンにセグメントを送信するスクリプトについては、Scorekeep サンプルアプリケーションを参照してください。

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

の使用 PutTraceSegments

セグメントドキュメントは、PutTraceSegments API を使用してアップロードできます。API には、単一のパラメーター (TraceSegmentDocuments) があり、これを実行すると、JSON セグメントドキュメントのリストが取得されます。

AWS CLI では、aws xray put-trace-segments コマンドを使用してセグメントドキュメントを直接 X-Ray に送信します。

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

> **Note**
>
> Windows コマンドプロセッサと Windows PowerShell では、JSON 文字列の引用符とエスケープ引用符の要件が異なります。詳細については、ユーザーガイドの「文字列の引用 AWS CLI 」を参照してください。

この出力には、処理に失敗したセグメントが表示されます。たとえば、トレース ID の日付が遠い過去の場合は、次のような エラーが表示されます。

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

複数のセグメントドキュメントは、スペースで区切って同時に渡すことができます。

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

セグメントドキュメントを X-Ray デーモンに送信する

セグメントドキュメントを X-Ray API に送信する代わりに、セグメントおよびサブセグメントを X-Ray デーモンに送信できます。これにより、バッファされた後、バッチで X-Ray API にアップロードされます。X-Ray SDK は、セグメントドキュメントをデーモンに送信して、 AWS が直接呼び出されないようにします。

> **Note**
>
> デーモンを実行する方法については、「ローカルで X-Ray; デーモンを実行する」を参照してください。

UDP ポート 2000 経由で JSON でセグメントを送信します。先頭にはデーモンのヘッダー `{"format": "json", "version": 1}\n` を追加します。

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

Linux では、セグメントドキュメントを Bash ターミナルからデーモンに送信できます。ヘッダーおよびセグメントドキュメントをテキストファイルに保存し、`cat` を使用して `/dev/udp` にパイプします。

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

Example segment.txt

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

デーモンのログを確認し、セグメントが X-Ray に送信されていることを確認します。

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

X-Ray からデータを取得する

X-Ray は、送信するトレースデータを処理して、JSON で完全なトレース、トレースの概要、サービスグラフを生成します。生成されたデータは、 AWS CLI を使用して API から直接取得できます。

サービスグラフの取得

JSON サービスグラフを取得するには、GetServiceGraph API を使用することができます。この API には、開始時間と終了時間を設定する必要があります。これらは date コマンドを使用して Linux 端末から計算することができます。

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

date +%s は、日付を秒単位で出力します。この数字を終了時間として使用し、日付から差し引いて、開始時間を取得します。

Example 最後の 10 分間のサービスグラフを取得するスクリプト。

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

次の例では、4 つのノードを持つサービスグラフを示します。これには、クライアントノード、EC2 インスタンス、DynamoDB テーブル、および Amazon SNS トピックが含まれます。

Example GetServiceGraph 出力

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

```
                "StartTime": 1528317567.0,
                "EndTime": 1528317589.0,
                "SummaryStatistics": {
                    "OkCount": 3,
                    "ErrorStatistics": {
                        "ThrottleCount": 0,
                        "OtherCount": 1,
                        "TotalCount": 1
                    },
                    "FaultStatistics": {
                        "OtherCount": 0,
                        "TotalCount": 0
                    },
                    "TotalCount": 4,
                    "TotalResponseTime": 0.273
                },
                "ResponseTimeHistogram": [
                    {
                        "Value": 0.005,
                        "Count": 1
                    },
                    {
                        "Value": 0.015,
                        "Count": 1
                    },
                    {
                        "Value": 0.157,
                        "Count": 1
                    },
                    {
                        "Value": 0.096,
                        "Count": 1
                    }
                ],
                "Aliases": []
            }
        ]
    },
    {
        "ReferenceId": 1,
        "Name": "awseb-e-dixzws4s9p-stack-StartupSignupsTable-4IMSMHAYX2BA",
        "Names": [
            "awseb-e-dixzws4s9p-stack-StartupSignupsTable-4IMSMHAYX2BA"
        ],
```

```
            "Type": "AWS::DynamoDB::Table",
            "State": "unknown",
            "StartTime": 1528317583.0,
            "EndTime": 1528317589.0,
            "Edges": [],
            "SummaryStatistics": {
                "OkCount": 2,
                "ErrorStatistics": {
                    "ThrottleCount": 0,
                    "OtherCount": 0,
                    "TotalCount": 0
                },
                "FaultStatistics": {
                    "OtherCount": 0,
                    "TotalCount": 0
                },
                "TotalCount": 2,
                "TotalResponseTime": 0.12
            },
            "DurationHistogram": [
                {
                    "Value": 0.076,
                    "Count": 1
                },
                {
                    "Value": 0.044,
                    "Count": 1
                }
            ],
            "ResponseTimeHistogram": [
                {
                    "Value": 0.076,
                    "Count": 1
                },
                {
                    "Value": 0.044,
                    "Count": 1
                }
            ]
        },
        {
            "ReferenceId": 2,
            "Name": "xray-sample.elasticbeanstalk.com",
            "Names": [
```

```
                "xray-sample.elasticbeanstalk.com"
            ],
            "Root": true,
            "Type": "AWS::EC2::Instance",
            "State": "active",
            "StartTime": 1528317567.0,
            "EndTime": 1528317589.0,
            "Edges": [
                {
                    "ReferenceId": 1,
                    "StartTime": 1528317567.0,
                    "EndTime": 1528317589.0,
                    "SummaryStatistics": {
                        "OkCount": 2,
                        "ErrorStatistics": {
                            "ThrottleCount": 0,
                            "OtherCount": 0,
                            "TotalCount": 0
                        },
                        "FaultStatistics": {
                            "OtherCount": 0,
                            "TotalCount": 0
                        },
                        "TotalCount": 2,
                        "TotalResponseTime": 0.12
                    },
                    "ResponseTimeHistogram": [
                        {
                            "Value": 0.076,
                            "Count": 1
                        },
                        {
                            "Value": 0.044,
                            "Count": 1
                        }
                    ],
                    "Aliases": []
                },
                {
                    "ReferenceId": 3,
                    "StartTime": 1528317567.0,
                    "EndTime": 1528317589.0,
                    "SummaryStatistics": {
                        "OkCount": 2,
```

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

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

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

グループ別サービスグラフの取得

グループの内容に基づきサービスグラフを呼び出すには、groupName または groupARN を含めます。以下の例では、Example1 という名前のグループへのサービスグラフの呼び出しを示します。

Example グループ Example1 の名前別サービスグラフを取得するスクリプト

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

トレースの取得

GetTraceSummaries API を使用して、トレースサマリのリストを取得します。トレースサマリには、ダウンロードするトレース全体 (注釈、リクエストと応答に関する情報、ID) などを識別するのに使用できる情報が含まれます。

`aws xray get-trace-summaries` を呼び出すときに、2 つの TimeRangeType フラグを使用できます。

- TraceId – デフォルトのGetTraceSummaries検索では TraceID 時間を使用し、計算[start_time, end_time)範囲内で開始されたトレースを返します。このタイムスタンプの範囲は、 内のタイムスタンプのエンコードに基づいて計算されるか TraceId、手動で定義できます。
- イベント時間 – 時間の経過とともに発生したイベントを検索するために、 AWS X-Ray ではイベントタイムスタンプを使用してトレースを検索できます。イベント時間では、トレースの開始時間に関係なく、[start_time, end_time) の範囲内でアクティブなトレースが返されます。

`aws xray get-trace-summaries` コマンドを使用して、トレースサマリのリストを取得します。次のコマンドは、デフォルト TraceId 時間を使用して、過去 1~2 分間のトレースの概要のリストを取得します。

Example トレースサマリを取得するスクリプト

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

Example GetTraceSummaries 出力

```
{
    "TraceSummaries": [
        {
            "HasError": false,
            "Http": {
                "HttpStatus": 200,
                "ClientIp": "205.255.255.183",
                "HttpURL": "http://scorekeep.elasticbeanstalk.com/api/session",
```

```
                "UserAgent": "Mozilla/5.0 (Windows NT 6.1; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.115 Safari/537.36",
                "HttpMethod": "POST"
            },
            "Users": [],
            "HasFault": false,
            "Annotations": {},
            "ResponseTime": 0.084,
            "Duration": 0.084,
            "Id": "1-59602606-a43a1ac52fc7ee0eea12a82c",
            "HasThrottle": false
        },
        {
            "HasError": false,
            "Http": {
                "HttpStatus": 200,
                "ClientIp": "205.255.255.183",
                "HttpURL": "http://scorekeep.elasticbeanstalk.com/api/user",
                "UserAgent": "Mozilla/5.0 (Windows NT 6.1; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.115 Safari/537.36",
                "HttpMethod": "POST"
            },
            "Users": [
                {
                    "UserName": "5M388M1E"
                }
            ],
            "HasFault": false,
            "Annotations": {
                "UserID": [
                    {
                        "AnnotationValue": {
                            "StringValue": "5M388M1E"
                        }
                    }
                ],
                "Name": [
                    {
                        "AnnotationValue": {
                            "StringValue": "Ola"
                        }
                    }
                ]
            },
```

```
            "ResponseTime": 3.232,
            "Duration": 3.232,
            "Id": "1-59602603-23fc5b688855d396af79b496",
            "HasThrottle": false
        }
    ],
    "ApproximateTime": 1499473304.0,
    "TracesProcessedCount": 2
}
```

出力のトレース ID を使用して、BatchGetTraces API でトレース全体を取得します。

Example BatchGetTraces コマンド

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

Example BatchGetTraces 出力

```
{
    "Traces": [
        {
            "Duration": 3.232,
            "Segments": [
                {
                    "Document": "{\"id\":\"1fb07842d944e714\",\"name\":
\"random-name\",\"start_time\":1.499473411677E9,\"end_time\":1.499473414572E9,
\"parent_id\":\"0c544c1b1bbff948\",\"http\":{\"response\":{\"status\":200}},
\"aws\":{\"request_id\":\"ac086670-6373-11e7-a174-f31b3397f190\"},\"trace_id\":
\"1-59602603-23fc5b688855d396af79b496\",\"origin\":\"AWS::Lambda\",\"resource_arn\":
\"arn:aws:lambda:us-west-2:123456789012:function:random-name\"}",
                    "Id": "1fb07842d944e714"
                },
                {
                    "Document": "{\"id\":\"194fcc8747581230\",\"name\":\"Scorekeep
\",\"start_time\":1.499473411562E9,\"end_time\":1.499473414794E9,\"http\":{\"request
\":{\"url\":\"http://scorekeep.elasticbeanstalk.com/api/user\",\"method\":\"POST\",
\"user_agent\":\"Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML,
 like Gecko) Chrome/59.0.3071.115 Safari/537.36\",\"client_ip\":\"205.251.233.183\"},
\"response\":{\"status\":200}},\"aws\":{\"elastic_beanstalk\":{\"version_label\":\"app-
abb9-170708_002045\",\"deployment_id\":406,\"environment_name\":\"scorekeep-dev\"},
\"ec2\":{\"availability_zone\":\"us-west-2c\",\"instance_id\":\"i-0cd9e448944061b4a
\"},\"xray\":{\"sdk_version\":\"1.1.2\",\"sdk\":\"X-Ray for Java\"}},\"service
\":{},\"trace_id\":\"1-59602603-23fc5b688855d396af79b496\",\"user\":\"5M388M1E
```

\",\"origin\":\"AWS::ElasticBeanstalk::Environment\",\"subsegments\":[{\"id\":
\"0c544c1b1bbff948\",\"name\":\"Lambda\",\"start_time\":1.499473411629E9,\"end_time
\":1.499473414572E9,\"http\":{\"response\":{\"status\":200,\"content_length\":14}},
\"aws\":{\"log_type\":\"None\",\"status_code\":200,\"function_name\":\"random-name
\",\"invocation_type\":\"RequestResponse\",\"operation\":\"Invoke\",\"request_id
\":\"ac086670-6373-11e7-a174-f31b3397f190\",\"resource_names\":[\"random-name\"]},
\"namespace\":\"aws\"},{\"id\":\"071684f2e555e571\",\"name\":\"## UserModel.saveUser
\",\"start_time\":1.499473414581E9,\"end_time\":1.499473414769E9,\"metadata\":{\"debug
\":{\"test\":\"Metadata string from UserModel.saveUser\"}},\"subsegments\":[{\"id\":
\"4cd3f10b76c624b4\",\"name\":\"DynamoDB\",\"start_time\":1.49947341469E9,\"end_time
\":1.499473414769E9,\"http\":{\"response\":{\"status\":200,\"content_length\":57}},
\"aws\":{\"table_name\":\"scorekeep-user\",\"operation\":\"UpdateItem\",\"request_id
\":\"MFQ8CGJ3JTDDVVVASUAAJGQ6NJ82F738BOB4KQNSO5AEMVJF66Q9\",\"resource_names\":
[\"scorekeep-user\"]},\"namespace\":\"aws\"}]}]}",
                    "Id": "194fcc8747581230"
                },
                {
                    "Document": "{\"id\":\"00f91aa01f4984fd\",\"name\":
\"random-name\",\"start_time\":1.49947341283E9,\"end_time\":1.49947341457E9,
\"parent_id\":\"1fb07842d944e714\",\"aws\":{\"function_arn\":\"arn:aws:lambda:us-
west-2:123456789012:function:random-name\",\"resource_names\":[\"random-name\"],
\"account_id\":\"123456789012\"},\"trace_id\":\"1-59602603-23fc5b688855d396af79b496\",
\"origin\":\"AWS::Lambda::Function\",\"subsegments\":[{\"id\":\"e6d2fe619f827804\",
\"name\":\"annotations\",\"start_time\":1.499473413012E9,\"end_time\":1.499473413069E9,
\"annotations\":{\"UserID\":\"5M388M1E\",\"Name\":\"Ola\"}},{\"id\":\"b29b548af4d54a0f
\",\"name\":\"SNS\",\"start_time\":1.499473413112E9,\"end_time\":1.499473414071E9,
\"http\":{\"response\":{\"status\":200}},\"aws\":{\"operation\":\"Publish\",
\"region\":\"us-west-2\",\"request_id\":\"a2137970-f6fc-5029-83e8-28aadeb99198\",
\"retries\":0,\"topic_arn\":\"arn:aws:sns:us-west-2:123456789012:awseb-e-
ruag3jyweb-stack-NotificationTopic-6B829NT9V5O9\"},\"namespace\":\"aws\"},{\"id\":
\"2279c0030c955e52\",\"name\":\"Initialization\",\"start_time\":1.499473412064E9,
\"end_time\":1.499473412819E9,\"aws\":{\"function_arn\":\"arn:aws:lambda:us-
west-2:123456789012:function:random-name\"}}]}",
                    "Id": "00f91aa01f4984fd"
                },
                {
                    "Document": "{\"id\":\"17ba309b32c7fbaf\",\"name\":
\"DynamoDB\",\"start_time\":1.49947341469E9,\"end_time\":1.499473414769E9,
\"parent_id\":\"4cd3f10b76c624b4\",\"inferred\":true,\"http\":{\"response
\":{\"status\":200,\"content_length\":57}},\"aws\":{\"table_name
\":\"scorekeep-user\",\"operation\":\"UpdateItem\",\"request_id\":
\"MFQ8CGJ3JTDDVVVASUAAJGQ6NJ82F738BOB4KQNSO5AEMVJF66Q9\",\"resource_names\":
[\"scorekeep-user\"]},\"trace_id\":\"1-59602603-23fc5b688855d396af79b496\",\"origin\":
\"AWS::DynamoDB::Table\"}",

```
                    "Id": "17ba309b32c7fbaf"
                },
                {
                    "Document": "{\"id\":\"1ee3c4a523f89ca5\",\"name\":\"SNS
\",\"start_time\":1.499473413112E9,\"end_time\":1.499473414071E9,\"parent_id\":
\"b29b548af4d54a0f\",\"inferred\":true,\"http\":{\"response\":{\"status\":200}},\"aws
\":{\"operation\":\"Publish\",\"region\":\"us-west-2\",\"request_id\":\"a2137970-
f6fc-5029-83e8-28aadeb99198\",\"retries\":0,\"topic_arn\":\"arn:aws:sns:us-
west-2:123456789012:awseb-e-ruag3jyweb-stack-NotificationTopic-6B829NT9V5O9\"},
\"trace_id\":\"1-59602603-23fc5b688855d396af79b496\",\"origin\":\"AWS::SNS\"}",
                    "Id": "1ee3c4a523f89ca5"
                }
            ],
            "Id": "1-59602603-23fc5b688855d396af79b496"
        }
    ],
    "UnprocessedTraceIds": []
}
```

トレース全体には、同一のトレース ID を使用して取得されるすべてのセグメントドキュメントからコンパイルされた、各セグメントのドキュメントが含まれます。これらのドキュメントは、アプリケーションによって X-Ray に送信されたデータを表していません。その代わりに、X-Ray サービスによって生成された処理済みドキュメントを表します。X-Ray はアプリケーションによって送信されたセグメントドキュメントをコンパイルして、完全なトレースドキュメントを作成し、セグメントドキュメントスキーマに準拠しないデータを削除します。詳細については、「X-Ray セグメントドキュメント」を参照してください。

X-Ray は、セグメント自体を送信しないサービスへのダウンストリーム呼び出しの推測セグメントを作成します。たとえば、計測されたクライアントを使用して DynamoDB を呼び出したときに、X-Ray SDK の視点からの呼び出しに関する詳細をサブセグメントに記録します。ただし、DynamoDB は対応するセグメントを送信しません。X-Ray は、サブセグメントの情報を使用して、トレースマップ内の DynamoDB リソースを表す推定セグメントを作成し、トレースドキュメントに追加します。

API から複数のトレースを取得するには、`get-trace-summaries` の出力から AWS CLI クエリを使用して抽出できるトレース ID のリストが必要です。リストから `batch-get-traces` の入力にリダイレクトし、特定の時間のトレース全体を取得します。

Example 1 分間のトレース全体を取得するスクリプト。

```
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[*]'
```

根本原因分析の取得と絞り込み

GetTraceSummaries API を使用してトレースサマリーを生成すると、部分的なトレースサマリーを JSON 形式で再利用して、根本原因に基づいて絞り込まれたフィルター式を作成できます。絞り込みのステップのウォークスルーについては、以下の例を参照してください。

Example GetTraceSummaries 出力例 - 応答時間の根本原因セクション

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

上記の出力を編集して省略することで、この JSON は一致した根本原因のエンティティのフィルタになる可能性があります。JSON に存在するすべてのフィールドについて、候補は完全に一致する必要があります。そうしないと、トレースが返されません。削除されたフィールドはワイルドカード値になります。これは、フィルタ式クエリ構造と互換性のある形式です。

Example 再フォーマットされた応答時間の根本原因

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

この JSON は、`rootcause.json = #[{}]` への呼び出しを通じてフィルタ式の一部として使用されます。フィルター式を使用したクエリの詳細については、「X-Ray コンソールの探索」の「フィルター式の使用」セクションを参照してください。

Example JSON フィルタの例

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

X-Ray API を使用したサンプリング、グループ、および暗号化設定の構成

X-Ray にはAPIs が用意されています。

暗号化設定

PutEncryptionConfig を使用して、暗号化に使用する AWS Key Management Service （ AWS KMS) キーを指定します。

> **Note**
>
> X-Ray は非対称 KMS キーをサポートしていません。

```
$ 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"
    }
}
```

キー ID には、(例に示すような) エイリアス、キー ID、または Amazon リソースネーム (ARN) を使用できます。

GetEncryptionConfigを使用して現在の設定を取得します。X-Ray が設定の適用を終了すると、ステータスが [UPDATING] から [ACTIVE] に変わります。

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

```
        "Status": "ACTIVE",
        "Type": "KMS"
    }
}
```

KMS の使用を停止し、デフォルトの暗号化を使用するには、暗号化タイプを NONE に設定します。

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

サンプリングルール

X-Ray API を使用して、アカウントのサンプリングルールを管理できます。サンプリングの詳細については、「」を参照してくださいサンプリングルールを設定する。タグの追加と管理の詳細については、「X-Ray のサンプリングルールとグループのタグ付け」を参照してください。

GetSamplingRulesですべてのサンプリングルールを取得します。

```
$ 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
        }
    ]
}
```

別のルールに一致しないすべてのリクエストにデフォルトのルールが適用されます。このルールは最も優先度が低く、削除することはできません。ただし、UpdateSamplingRuleを使用してレートとリザーバのサイズを変更できます。

Example **UpdateSamplingRule**の API 入力 – 10000-default.json

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

次の例では、以前のファイルを入力として使用し、デフォルトのルールをリザーバなしの 1% に変更します。タグはオプションです。タグを追加する場合は、タグキーが必要で、タグ値はオプションです。サンプリングルールから既存のタグを削除するには、 を使用します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
        },
```

CreateSamplingRuleを使用して追加のサンプリングルールを作成します。ルールを作成するときは、ルールフィールドの大部分を指定する必要があります。次の例では 2 つのルールを作成します。この最初のルールでは、Scorekeep サンプルアプリケーションの基本レートを設定します。これは、より優先度の高いルールに一致しない API からのすべてのリクエストに一致します。

Example **UpdateSamplingRule**の API 入力 – 9000-base-scorekeep.json

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

2 つ目のルールも Scorekeep に適用されますが、このルールはより優先度が高く具体的です。このルールは、ポーリングリクエストに関して非常に低いサンプリングレートを設定します。これらは、ゲームの状態の変更を確認するためにクライアントによって数秒ごとに行われる GET リクエストです。

Example **UpdateSamplingRule**の API 入力 – 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
    }
}
```

タグはオプションです。タグを追加する場合は、タグキーが必要で、タグ値はオプションです。

```
$ 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
    }
}
```

サンプリングルールを削除するには、DeleteSamplingRuleを使用します。

```
$ 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
    }
}
```

グループ

X-Ray API を使用して、アカウントのグループを管理することができます。グループは、フィルタ式で定義されるトレースのコレクションです。グループを使用して追加のサービスグラフを生成し、Amazon CloudWatch メトリクスを指定できます。X-Ray API を使用したサービスグラフとメトリクスの操作の詳細については、「X-Ray からデータを取得する」を参照してください。グループの詳細については、「グループを設定する」を参照してください。タグの追加と管理の詳細については、「X-Ray のサンプリングルールとグループのタグ付け」を参照してください。

`CreateGroup` を使用してグループを作成します。タグはオプションです。タグを追加する場合は、タグキーが必要で、タグ値はオプションです。

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

`GetGroups` を使用して既存のグループをすべて取得します。

```
$ 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"
}
```

UpdateGroup を使用してグループを更新します。タグはオプションです。タグを追加する場合は、タグキーが必要で、タグ値はオプションです。グループから既存のタグを削除するには、 を使用します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}"
}
```

DeleteGroup を使用してグループを削除します。

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

X-Ray API でのサンプリングルールの使用

X-Ray SDK は X-Ray API を使用して、サンプリングルールの取得、サンプリング結果のレポート、クォータの取得を行います。これらの API を使用すれば、サンプリングルールの仕組みを理解したり、X-Ray SDK でサポートされていない言語でサンプリングを実行したりできます。

まず、GetSamplingRulesを使用してすべてのサンプリングルールを取得します。

```
$ 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
        }
    ]
}
```

出力には、デフォルトルールとカスタムルールが含まれています。まだサンプリングルールを作成していない場合は、「X-Ray API を使用したサンプリング、グループ、および暗号化設定の構成」を参照してください。

優先度の昇順で受信リクエストのルールを評価します。ルールが一致したら、固定レートとリザーバのサイズを使用してサンプリングデシジョンを作成します。サンプリングされたリクエストを記録し、(トレースを目的とする) サンプリングされていないリクエストは無視します。サンプリングデシジョンが作成されたら、ルールの評価を停止します。

ルールのリザーバのサイズは、固定レートを適用する前に記録する 1 秒あたりのトレースの目標数です。リザーバはすべてのサービスに累積的に適用されるため、直接使用することはできません。ただし、0 以外の場合は、X-Ray がクォータを割り当てるまでリザーバから 1 秒に 1 個トレースを借りることが可能です。クォータを受信する前に、1 秒ごとに最初のリクエストを記録し、追加のリクエストに固定レートを適用します。固定レートは、0 ~ 1.00 (100%) の 10 進数です。

次の例は、過去 10 秒間に作成されたサンプリングデシジョンの詳細を含むGetSamplingTargetsの呼び出しを示したものです。

```
$ aws xray get-sampling-targets --sampling-statistics-documents '[
    {
        "RuleName": "base-scorekeep",
        "ClientID": "ABCDEF1234567890ABCDEF10",
        "Timestamp": "2018-07-07T00:20:06",
        "RequestCount": 110,
        "SampledCount": 20,
        "BorrowCount": 10
    },
    {
        "RuleName": "polling-scorekeep",
        "ClientID": "ABCDEF1234567890ABCDEF10",
        "Timestamp": "2018-07-07T00:20:06",
        "RequestCount": 10500,
        "SampledCount": 31,
        "BorrowCount": 0
```

```
    }
]'
{
    "SamplingTargetDocuments": [
        {
            "RuleName": "base-scorekeep",
            "FixedRate": 0.1,
            "ReservoirQuota": 2,
            "ReservoirQuotaTTL": 1530923107.0,
            "Interval": 10
        },
        {
            "RuleName": "polling-scorekeep",
            "FixedRate": 0.003,
            "ReservoirQuota": 0,
            "ReservoirQuotaTTL": 1530923107.0,
            "Interval": 10
        }
    ],
    "LastRuleModification": 1530920505.0,
    "UnprocessedStatistics": []
}
```

X-Ray からのレスポンスには、リザーバから借りる代わりに使用するクォータが含まれています。この例では、サービスがリザーバから 10 秒間に 10 個のトレースを借り、他の 100 個のリクエストに 10% の固定レートを適用した結果、サンプリングされたリクエストの合計数が 20 個になりました。クォータは (有効期限で示される) 5 分間、または新しいクォータが割り当てられるまで有効です。X-Ray では、ここで割り当てられなかったとしても、デフォルトより長いレポート間隔を割り当てる場合があります。

> **Note**
>
> 最初の呼び出しのときには、X-Ray からのレスポンスにクォータが含まれていない可能性があります。その場合は、クォータが割り当てられるまで、リザーバからクォータを借り続けてください。

レスポンスの他の 2 つのフィールドは、入力の問題を示している可能性があるため、前回呼び出した`GetSamplingRules`の `LastRuleModification` を確認します。より新しい場合は、そのルールの新しいコピーを取得します。`UnprocessedStatistics` には、ルールが削除されたこと、入力

の統計ドキュメントが古すぎること、またはアクセス許可のエラーが発生していることを示すエラーが含まれている可能性があります。

X-Ray セグメントドキュメント

トレースセグメントは、アプリケーションが対応するリクエストの JSON 表現です。トレースセグメントは、元のリクエストに関する情報、アプリケーションがローカルで実行する作業に関する情報、およびアプリケーションが リソース、HTTP API、SQL データベースに対して行うダウンストリーム呼び出しに関する情報のサブセグメント AWS を記録します。

セグメントドキュメントは、セグメントに関する情報を X-Ray に伝えます。セグメントドキュメントは最大で 64 kB とし、サブセグメントを含むセグメント全体、リクエストが進行中であることを示すセグメントのフラグメント、または別個に送信される単一のサブセグメントを含むことができます。セグメントドキュメントは、`PutTraceSegments` API を使用して直接 X-Ray に送信できます。

X-Ray はセグメントドキュメントをコンパイルおよび処理し、それぞれ`GetTraceSummaries`および`BatchGetTraces` API を使用してアクセスできる、クエリ可能なトレースサマリおよびトレース全体を生成します。このサービスは、X-Ray に送信するセグメントとサブセグメントに加えて、サブセグメントの情報を使用して推測セグメントを生成し、トレース全体に追加します。推定セグメントは、トレースマップ内のダウンストリームサービスとリソースを表します。

X-Ray は、セグメントドキュメントの JSON スキーマを提供します。スキーマは xray-segmentdocument-schema-v1.0.0 からダウンロードできます。スキーマに示されたフィールドとオブジェクトについては、以下のセクションで詳しく説明します。

セグメントフィールドのサブセットは、フィルタ式で使用するために X-Ray によってインデックスが作成されます。たとえば、セグメントの `user` フィールドを一意の ID に設定した場合、X-Ray コンソールで、または `GetTraceSummaries` API を使用して、特定のユーザーに関連付けられたセグメントを検索できます。詳細については、「フィルター式を使用する」を参照してください。

X-Ray SDK でアプリケーションを計測すると、SDK によりセグメントドキュメントが生成されます。セグメントドキュメントを直接 X-Ray に送信する代わりに、SDK がそれらのドキュメントをローカル UDP ポート経由で X-Ray デーモンに送信します。詳細については、「セグメントドキュメントを X-Ray デーモンに送信する」を参照してください。

セグメントフィールド

セグメントは、アプリケーションが対応するリクエストに関する追跡情報を記録します。セグメントは、少なくともリクエストの名前、ID、開始時間、トレース ID、および終了時間を記録します。

Example 最小完了セグメント

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

次のフィールドは、セグメントで必須、または条件付きで必須です。

> **Note**
>
> 特に明記されていない限り、値は文字列である必要があります (最大 250 文字)。

必須のセグメントフィールド

- name – リクエストを処理したサービスの論理名 (最大 200 文字)。たとえば、アプリケーション名やドメイン名です。名前には、Unicode 文字、数字、空白、および次の記号を含めることができます: _、.、:、/、%、&、#、=、+、\、-、@
- id – セグメントの 64 ビット識別子。16 進数の数字であり、同じトレース内のセグメント間で一意です。
- trace_id – 1 つのクライアントリクエストから送信されるすべてのセグメントとサブセグメントに接続する一意の識別子です。

  X-Ray トレース ID 形式

  X-Ray trace_id は、ハイフンで区切られた 3 つの数字で構成されています。例えば、1-58406520-a006649127e371903a2de979 と指定します。これには、以下のものが含まれます :

  - バージョン番号。 1
  - 8 桁の 16 進数を使用した Unix エポック時間での元のリクエストの時刻。

    例えば、2016 年 12 月 1 日午前 10:00 PST のエポックタイムは1480615200秒、16 58406520 進数です。
  - 24 桁の 16 進数のトレースのグローバルに一意の 96 ビット識別子。

> **Note**
>
> X-Ray は、 OpenTelemetry および W3C トレースIDs をサポートするようになりました。 W3C W3C トレース ID は、X-Ray に送信するときに X-Ray トレース ID 形式でフォーマットする必要があります。例えば、W3C トレース ID は X-Ray に送信する`1-4efaaf4d-1e8720b39541901950019ee5`ときに としてフォーマット`4efaaf4d1e8720b39541901950019ee5`する必要があります。X-Ray トレース IDs には、Unix エポックタイムの元のリクエストタイムスタンプが含まれますが、これは W3C トレース IDs X-Ray 形式で送信する場合には必要ありません。

> **トレース ID セキュリティ**
>
> トレース ID はレスポンスヘッダーに表示されます。攻撃者が将来のトレース ID を計算できないように安全なランダムのアルゴリズムを使用してトレース ID を生成し、その ID を使用してアプリケーションにリクエストを送信します。

- `start_time` – セグメントが作成された時間の数値 (エポック時間の浮動小数点で表した秒数)。例えば、`1480615200.010`、`1.480615200010E9` などです。必要なだけ桁数を使用します。利用できる場合は、マイクロ秒の精度をお勧めします。
- `end_time` – セグメントが切断された時間を表す数値。例えば、`1480615200.090`、`1.480615200090E9` などです。`end_time` または `in_progress` のどちらかを指定します。
- `in_progress` – ではなく を設定して、開始されたが完了していないセグメントを記録する`true`ブール値`end_time`。アプリケーションが処理に時間がかかるリクエストを受信したときに、進行中のセグメントを送信して、リクエストの受信を追跡します。レスポンスが送信されると、完了したセグメントが送信され進行中のセグメントを上書きします。リクエストごとに、1 つの完全なセグメントと、1 つまたは 0 個の進行中のセグメントのみを送信します。

> **サービス名**
>
> セグメントの `name` は、セグメントを生成するサービスのドメイン名または論理名と一致する必要があります。ただし、これは強制ではありません。権限を持つアプリケーション`PutTraceSegments`は、任意の名前でセグメントを送信できます。

次のフィールドは、セグメントではオプションです。

オプションのセグメントフィールド

- `service` – アプリケーションに関する情報を含むオブジェクト。
  - `version` – リクエストに対応したアプリケーションのバージョンを識別する文字列。
- `user` – リクエストを送信したユーザーを識別する文字列。
- `origin` – アプリケーションを実行している AWS リソースのタイプ。

  サポートされる値

  - `AWS::EC2::Instance` – Amazon EC2 インスタンス。
  - `AWS::ECS::Container` – Amazon ECS コンテナ。
  - `AWS::ElasticBeanstalk::Environment` – Elastic Beanstalk 環境

  複数の値をアプリケーションに適用する場合は、最も具体的な値を使用します。たとえば、複数コンテナの Docker Elastic Beanstalk の環境では、Amazon ECS コンテナでアプリケーションが実行され、そのコンテナは Amazon EC2 インスタンスで実行されます。この場合、環境は他の 2 つのリソースの親として、オリジンを `AWS::ElasticBeanstalk::Environment` に設定します。
- `parent_id` – 計測したアプリケーションからリクエストが発信された場合に指定するサブセグメント ID。X-Ray SDK は親サブセグメント ID をダウンストリーム HTTP 呼び出しのトレースヘッダーに追加します。ネストされたサブセグメントの場合、サブセグメントは親としてセグメントまたはサブセグメントを持つことができます。
- `http` – 元の HTTP リクエストに関する情報を含む`http`オブジェクト。
- `aws` – アプリケーションがリクエストを処理した AWS リソースに関する情報を含む `aws` オブジェクト。
- `error`、`throttle`、`fault`、`cause` – エラーが発生したことを示し、エラーの原因となった例外に関する情報を含む `error` フィールド。
- `annotations` – X-Ray で検索用にインデックスを作成するキーと値のペアを含む`annotations`オブジェクト。
- `metadata` – セグメントに保存する追加のデータを含む`metadata`オブジェクト。
- `subsegments` – オブジェクトの配列`subsegment`。

サブセグメント

サブセグメントを作成して、 AWS SDK で行った AWS のサービス およびリソースへの呼び出し、内部または外部の HTTP ウェブ APIsへの呼び出し、または SQL データベースクエリを記録できます。また、サブセグメントを作成してアプリケーションでコードブロックをデバッグしたり、注釈を付けたりできます。サブセグメントには他のサブセグメントを含めることができるため、内部関数呼び出しに関するメタデータを記録するカスタムサブセグメントには、他のカスタムサブセグメントおよびダウンストリーム呼び出し用のサブセグメントを含めることができます。

サブセグメントは、ダウンストリーム呼び出しを、それを呼び出したサービスの視点から記録します。X-Ray はサブセグメントを使用して、セグメントを送信しないダウンストリームサービスを識別し、そのエントリをサービスグラフに作成します。

サブセグメントはフルセグメントドキュメントに埋め込むことも、個別に送信することもできます。サブセグメントを個別に送信して、長期実行されているリクエストのダウンストリーム呼び出しを非同期でトレースしたり、セグメントドキュメントの最大サイズを超えないようにしたりできます。

Example 埋め込みサブセグメントを含むセグメント

独立したサブセグメントには、親セグメントを識別する `type` の `subsegment`、および `parent_id` があります。

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

長期間実行されるリクエストについては、進行中のセグメントを送信してリクエストが受信されたことを X-Ray に通知し、セグメントを個別に送信して追跡してから、元のリクエストを完了することができます。

Example 進行中セグメント

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

Example 独立したサブセグメント

独立したサブセグメントには、親セグメントを識別する `type` の `subsegment`、`trace_id`、および `parent_id` があります。

```
{
  "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
      }
  }
}
```

リクエストが完了したら、end_time とともに再送信してセグメントを閉じます。完了セグメントは進行中のセグメントを上書きします。

非同期ワークフローをトリガーした、完了したリクエストに対してサブセグメントを個別に送信することもできます。たとえば、ウェブ API は、ユーザーがリクエストした作業を開始する直前に OK 200 応答を返す場合があります。応答が送信されたらすぐに、完全なセグメントを X-Ray に送信し、それに続いて後で完了する作業のサブセグメントを送信できます。セグメントと同様に、サブセグメントフラグメントを送信して、サブセグメントが開始されたことを記録した後で、ダウンストリーム呼び出しが完了したら完全なサブセグメントでそれを上書きできます。

次のフィールドは、サブセグメントで必須、または条件付きで必須です。

> **Note**
>
> 特に明記されていない限り、値は文字列です (最大 250 文字)。

必須のサブセグメントフィールド

- id – サブセグメントの 64 ビット識別子。16 進数の数字であり、同じトレース内のセグメント間で一意です。

- name – サブセグメントの論理名。ダウンストリーム呼び出しの場合は、リソースまたはサービスを呼び出した後のサブセグメントの名前。カスタムサブセグメントの場合は、計測するコードの後にサブセグメントの名前を付けます (関数名など)。
- start_time – サブセグメントが作成された時間を表す数値で、エポック時間を浮動小数点で表した秒 (ミリ秒)。例えば、1480615200.010、1.480615200010E9 などです。
- end_time – サブセグメントが切断された時間を表す数値。例えば、1480615200.090、1.480615200090E9 などです。end_time または in_progress を指定します。
- in_progress – ではなく を設定して、開始されたが完了していないサブセグメントを記録するtrueブール値end_time。ダウンストリームリクエストごとに、1 つの完全なサブセグメントと、1 つまたは 0 個の進行中のサブセグメントのみを送信します。
- trace_id – サブセグメントの親セグメントのトレース ID。サブセグメントを個別に送信する場合にのみ必要です。

  X-Ray トレース ID 形式

  X-Ray trace_id は、ハイフンで区切られた 3 つの数字で構成されています。例えば、1-58406520-a006649127e371903a2de979 と指定します。これには、以下のものが含まれます :

  - バージョン番号。 1
  - 8 桁の 16 進数を使用した Unix エポック時間での元のリクエストの時刻。

    例えば、2016 年 12 月 1 日午前 10:00 PST のエポックタイムは1480615200秒、16 58406520 進数です。
  - 24 桁の 16 進数のトレースのグローバルに一意の 96 ビット識別子。

> **Note**
>
> X-Ray は、 OpenTelemetry および W3C トレースIDs をサポートするようになりました。 W3C W3C トレース ID は、X-Ray に送信するときに X-Ray トレース ID 形式でフォーマットする必要があります。例えば、W3C トレース ID は X-Ray に送信する1-4efaaf4d-1e8720b39541901950019ee5ときに としてフォーマット4efaaf4d1e8720b39541901950019ee5する必要があります。X-Ray トレース IDs には、Unix エポックタイムの元のリクエストタイムスタンプが含まれますが、これは W3C トレース IDs X-Ray 形式で送信する場合には必要ありません。

- parent_id – サブセグメントの親セグメントのセグメント ID。サブセグメントを個別に送信する場合にのみ必要です。ネストされたサブセグメントの場合、サブセグメントは親としてセグメントまたはサブセグメントを持つことができます。
- type – subsegment。サブセグメントを個別に送信する場合にのみ必要です。

次のフィールドは、サブセグメントではオプションです。

オプションのサブセグメントフィールド

- namespace – AWS SDK 呼び出しの場合は aws、他のダウンストリーム呼び出しの場合は remote。
- http – 送信 HTTP 呼び出しに関する情報を含むhttpオブジェクト。
- aws – アプリケーションが呼び出したダウンストリーム AWS リソースに関する情報を含む aws オブジェクト。
- error、throttle、fault、cause – エラーが発生したことを示し、エラーの原因となった例外に関する情報を含む error フィールド。
- annotations – X-Ray で検索用にインデックスを作成するキーと値のペアを含むannotationsオブジェクト。
- metadata – セグメントに保存する追加のデータを含むmetadataオブジェクト。
- subsegments – subsegmentオブジェクトの配列。
- precursor_ids – このサブセグメントの前に完了した同じ親を持つサブセグメントを識別するサブセグメント ID の配列。

HTTP リクエストデータ

HTTP ブロックを使用して、(セグメントで) アプリケーションが対応した HTTP リクエスト、または (サブセグメントで) アプリケーションがダウンストリーム HTTP API に対して行ったリクエストの詳細を記録します。このオブジェクトのほとんどのフィールドは、HTTP リクエストと応答で見つかった情報にマッピングされます。

http

すべてのフィールドはオプションです。

- request – リクエストに関する情報。
  - method – リクエストメソッド。例えば GET です。

  - `url` – リクエストのプロトコル、ホスト名、およびパスからコンパイルされた、リクエストの完全な URL。
  - `user_agent` – リクエスタのクライアントからのユーザーエージェント文字列。
  - `client_ip` – リクエスタの IP アドレス。IP パケットの `Source Address` から、または転送リクエストの場合は `X-Forwarded-For` ヘッダーから取得できます。
  - `x_forwarded_for` – (セグメントのみ) が ヘッダーから読み取られ、偽造されている可能性があるため信頼できないことを示すclient_ipブール値X-Forwarded-For。
  - `traced` – (サブセグメントのみ) ダウンストリーム呼び出しが別の追跡されたサービスであることを示すブール値。このフィールドが `true` に設定されている場合、このブロックを含むサブセグメントの `parent_id` に一致する `id` を含むセグメントをダウンストリームサービスがアップロードするまで、X-Ray はトレースが壊れていると見なします。
- `response` – レスポンスに関する情報。
  - `status` – レスポンスの HTTP ステータスを示す整数。
  - `content_length` – レスポンス本文の長さをバイト単位で示す整数。

ダウンストリームウェブ API に対する呼び出しを計測するときは、HTTP リクエストおよびレスポンスに関する情報を含むセグメントを記録します。X-Ray はサブセグメントを使用してリモート API の推測セグメントを生成します。

Example Amazon EC2 で実行しているアプリケーションにより提供される HTTP 呼び出し用のセグメント

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

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

Example ダウンストリーム HTTP 呼び出しのサブセグメント

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

Example ダウンストリーム HTTP 呼び出しの推定セグメント

```
{
  "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
}
```

注釈

セグメントとサブセグメントは、X-Ray がフィルタ式で使用するためにインデックスを作成する 1 つ以上のフィールドが含まれた annotations オブジェクトを含むことができます。フィールドは、文字列、数値、またはブール値を持つことができます (オブジェクトや配列を含むことはできません)。X-Ray は、トレースごとに 50 の注釈までインデックスを付けます。

Example 注釈を使用した HTTP 呼び出しのセグメント

```
{
  "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
    }
  }
```

キーはフィルタで動作するために英数字である必要があります。アンダースコアは使用できます。その他の記号や空白は使用できません。

メタデータ

セグメントとサブセグメントは、オブジェクトと配列を含めて、任意の型の値を持つ 1 つ以上のフィールドが含まれた metadata オブジェクトを含むことができます。X-Ray はメタデータのインデックスを作成せず、セグメントドキュメントが最大サイズ (64 kB) を超えない限り、任意のサイズにすることができます。BatchGetTraces API によって返された完全なセグメントドキュメントで、メタデータを表示できます。で始まるフィールドキー (debug次の例では) AWS.は、 AWSが提供する SDKs とクライアントが使用するために予約されています。

Example カスタムサブセグメントとメタデータ

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

```
      "start_time": 1484789387.502,
      "end_time": 1484789387.534,
      "name": "DynamoDB",
      "namespace": "aws",
      "http": {
        "response": {
          "content_length": 58,
          "status": 200
        }
      },
      "aws": {
        "table_name": "scorekeep-user",
        "operation": "UpdateItem",
        "request_id": "3AIENM5J4ELQ3SPODHKBIRVIC3VV4KQNSO5AEMVJF66Q9ASUAAJG",
        "resource_names": [
          "scorekeep-user"
        ]
      }
    }
  ]
}
```

AWS リソースデータ

セグメントの場合、aws オブジェクトはアプリケーションが実行されているリソースに関する情報を含みます。複数のフィールドを単一のリソースに適用できます。たとえば、Elastic Beanstalk の複数コンテナの Docker 環境で実行されているアプリケーションは、Amazon EC2 インスタンス、インスタンス上で実行されている Amazon ECS コンテナ、および Elastic Beanstalk 環境自体に関する情報を持つことができます。

aws (セグメント)

すべてのフィールドはオプションです。

- account_id – アプリケーションが別の AWS アカウントにセグメントを送信する場合、アプリケーションを実行しているアカウントの ID を記録します。
- cloudwatch_logs – 単一の CloudWatch ロググループを記述するオブジェクトの配列。
  - log_group – CloudWatch ロググループ名。
  - arn – CloudWatch ロググループの ARN。
- ec2 – Amazon EC2 インスタンスに関する情報。
  - instance_id – EC2 インスタンスのインスタンス ID。

  - instance_size – EC2 インスタンスのタイプ。
  - ami_id – Amazon マシンイメージ ID。
  - availability_zone – インスタンスが実行されているアベイラビリティーゾーン。
- ecs – Amazon ECS コンテナに関する詳細。
  - container – コンテナのホスト名。
  - container_id – コンテナの完全なコンテナ ID。
  - container_arn – コンテナインスタンスの ARN。
- eks – Amazon EKS クラスターに関する情報。
  - pod – EKS ポッドのホスト名。
  - cluster_name – EKS クラスター名。
  - container_id – コンテナの完全なコンテナ ID。
- elastic_beanstalk – Elastic Beanstalk 環境に関する情報。この情報は、最新の Elastic Beanstalk プラットフォームの /var/elasticbeanstalk/xray/environment.conf という名前のファイルにあります。
  - environment_name – 環境の名前。
  - version_label – リクエストに対応したインスタンスに現在デプロイされているアプリケーションバージョンの名前。
  - deployment_id – リクエストに対応したインスタンスに対して最後に成功したデプロイの ID を示す数値。
- xray – 使用した計測のタイプとバージョンに関するメタデータ。
  - auto_instrumentation – 自動計測が使用されたかどうかを示すブール値 (例えば、Java Agent)。
  - sdk_version – 使用中の SDK またはエージェントのバージョン。
  - sdk – SDK のタイプ。

Example AWS プラグインを含む ブロック

```
"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"
   }
}
```

サブセグメントの場合は、アプリケーションがアクセスする AWS のサービス およびリソースに関する情報を記録します。X-Ray はこの情報を使用して、サービスマップのダウンストリームサービスを表す推定セグメントを作成します。

aws (サブセグメント)

すべてのフィールドはオプションです。

- operation – AWS のサービス またはリソースに対して呼び出された API アクションの名前。
- account_id – アプリケーションが別のアカウントのリソースにアクセスする場合、またはセグメントを別のアカウントに送信する場合は、アプリケーションがアクセスした AWS リソースを所有するアカウントの ID を記録します。
- region – リソースがアプリケーションとは異なるリージョンにある場合は、そのリージョンを記録します。例えば us-west-2 です。
- request_id – リクエストの一意の識別子。
- queue_url – Amazon SQS キューのオペレーションの場合は、キューの URL。
- table_name – DynamoDB テーブルのオペレーションの場合、テーブルの名前。

Example 項目を保存するための DynamoDB に対する呼び出しのサブセグメント

```
{
```

```
  "id": "24756640c0d0978a",
  "start_time": 1.480305974194E9,
  "end_time": 1.4803059742E9,
  "name": "DynamoDB",
  "namespace": "aws",
  "http": {
    "response": {
      "content_length": 60,
      "status": 200
    }
  },
  "aws": {
    "table_name": "scorekeep-user",
    "operation": "UpdateItem",
    "request_id": "UBQNSO5AEM8T4FDA4RQDEB94OVTDRVV4K4HIRGVJF66Q9ASUAAJG",
  }
}
```

エラーと例外

エラーが発生した場合は、エラーと生成された例外に関する詳細を記録できます。アプリケーションがユーザーにエラーを返す場合はセグメントにエラーを記録し、ダウンストリーム呼び出しがエラーを返す場合はサブセグメントにエラーを記録します。

エラーのタイプ

次の 1 つ以上のフィールドを `true` に設定して、エラーが発生したことを示します。複合エラーの場合は、複数のタイプを適用できます。たとえば、ダウンストリーム呼び出しからの `429 Too Many Requests` エラーにより、アプリケーションは `500 Internal Server Error` を返すことがあり、その場合は 3 つすべてのタイプが適用されます。

- `error` – クライアントエラーが発生したことを示すブール値 (レスポンスステータスコードは 4XX Client Error でした)。
- `throttle` – リクエストが調整されたことを示すブール値 (レスポンスステータスコードは 429 Too Many Requests でした)。
- `fault` – サーバーエラーが発生したことを示すブール値 (レスポンスステータスコードは 5XX Server Error でした)。

セグメントまたはサブセグメントに cause オブジェクトを含めてエラーの原因を示します。

cause

原因は、16 文字の例外 ID、または次のフィールドを含むオブジェクトとすることができます。

- working_directory – 例外が発生したときの作業ディレクトリのフルパス。
- paths – 例外が発生したときに使用されているライブラリまたはモジュールへのパスの配列。
- exceptions – 例外オブジェクトの配列。

1 つまたは複数の例外オブジェクトのエラーに関する詳細情報を含めます。

exception

すべてのフィールドはオプションです。

- id – 例外の 64 ビット識別子。16 進数の数字であり、同じトレース内のセグメント間で一意です。
- message – 例外メッセージ。
- type – 例外のタイプ。
- remote – ダウンストリームサービスによって返されたエラーが原因で例外が発生したことを示すブール値。
- truncated – から省略されたスタックフレームの数を示す整数stack。
- skipped – この例外とその子の間でスキップされた例外 (発生した例外) の数を示す整数。
- cause – 例外の親 (この例外を発生させた例外) の例外 ID。
- stack – stackFrame オブジェクトの配列。

使用可能な場合、コールスタックに関する情報を stackFrame オブジェクトに記録します。

stackFrame

すべてのフィールドはオプションです。

- path – ファイルの相対パス。
- line – ファイルの行。
- label – 関数またはメソッド名。

SQL クエリ

アプリケーションが SQL データベースに対して実行するクエリのサブセグメントを作成できます。

sql

すべてのフィールドはオプションです。

- connection_string – SQL Server または URL 接続文字列を使用しないその他のデータベース接続の場合は、パスワードを除く接続文字列を記録します。
- url – URL 接続文字列を使用するデータベース接続の場合は、パスワードを除く URL を記録します。
- sanitized_query – データベースクエリと、プレースホルダーによって削除または置換されたユーザー指定の値。
- database_type – データベースエンジンの名前。
- database_version – データベースエンジンのバージョン番号。
- driver_version – アプリケーションが使用するデータベースエンジンドライバーの名前とバージョン番号。
- user – データベースユーザー名。
- preparation – クエリで call を使用した場合は PreparedCall、クエリで statement を使用した場合は PreparedStatement。

Example サブセグメントと SQL クエリ

```
{
  "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 デーモン

> **Note**
>
> CloudWatch エージェントを使用して、Amazon EC2 インスタンスおよびオンプレミスサーバーからメトリクス、ログ、トレースを収集できるようになりました。 CloudWatch エージェントバージョン 1.300025.0 以降では、 OpenTelemetryまたは X-Ray クライアント SDKs からトレースを収集して X-Ray に送信できます。Distro for OpenTelemetry (ADOT) Collector AWS または X-Ray デーモンの代わりに CloudWatch エージェントを使用してトレースを収集することで、管理するエージェントの数を減らすことができます。詳細については、「 CloudWatch ユーザーガイド」のCloudWatch 「 エージェント」トピックを参照してください。

AWS X-Ray デーモンは、UDP ポート 2000 でトラフィックをリッスンし、raw セグメントデータを収集して API に中継するソフトウェアアプリケーションです AWS X-Ray 。デーモンは AWS X-Ray SDKs と連動し、SDK によって送信されたデータが X-Ray サービス SDKs に到達できるように実行されている必要があります。X-Ray デーモンは、オープンソースプロジェクトです。プロジェクトに従い、 で問題やプルリクエストを送信できます GitHub。github.com/aws/aws-xray-daemon

AWS Lambda および では AWS Elastic Beanstalk、これらのサービスの X-Ray との統合を使用してデーモンを実行します。Lambda は、サンプルリクエスト用に関数が呼び出される度に自動的にデーモンを実行します。Elastic Beanstalk では、`XRayEnabled`設定オプションを使用して、環境のインスタンスでデーモンを実行します。詳細については、以下を参照してください。

X-Ray デーモンをローカル、オンプレミス、またはその他の で実行するには AWS のサービス、ダウンロードして を実行し、セグメントドキュメントを X-Ray にアップロードするアクセス許可を付与します。

デーモンのダウンロード

デーモンは、Amazon S3、Amazon ECR、または Docker Hub からダウンロードしてローカルで実行するか、起動時に Amazon EC2 インスタンスにインストールします。

Amazon S3

X-Ray デーモンのインストーラおよび実行ファイル

- Linux (実行可能ファイル) – aws-xray-daemon-linux-3.x.zip (sig)
- Linux (RPM インストーラ) – aws-xray-daemon-3.x.rpm
- Linux (DEB インストーラ) – aws-xray-daemon-3.x.deb
- Linux (ARM64、実行可能ファイル) – aws-xray-daemon-linux-arm64-3.x.zip (sig)
- Linux (ARM64、RPM インストーラ) – aws-xray-daemon-arm64-3.x.rpm
- Linux (ARM64、DEB インストーラ) – aws-xray-daemon-arm64-3.x.deb
- OS X (実行可能ファイル) – aws-xray-daemon-macos-3.x.zip (sig)
- Windows (実行可能ファイル) – aws-xray-daemon-windows-process-3.x.zip (sig)
- Windows (サービス) – aws-xray-daemon-windows-service-3.x.zip (sig)

これらのリンクは、常にデーモンの最新の3.xリリースを指しています。特定のリリースをダウンロードするには、次の手順を実行します。

- バージョン より前のリリースをダウンロードする場合は3.3.0、 をバージョン番号3.xに置き換えます。例えば 2.1.0 です。バージョン より前のバージョンでは3.3.0、使用可能なアーキテクチャは のみですarm64。
- バージョン 以降のリリースをダウンロードする場合は3.3.0、 をバージョン番号3.xに、 をアーキテクチャタイプarchに置き換えます。例えば、2.1.0 と arm64 です。

X-Ray アセットは、サポートされている各リージョンのバケットにレプリケートされます。自分または AWS リソースに最も近いバケットを使用するには、上記のリンクのリージョンを自分のリージョンに置き換えます。

```
https://s3.us-west-2.amazonaws.com/aws-xray-assets.us-west-2/xray-daemon/aws-xray-daemon-3.x.rpm
```

Amazon ECR

バージョン 3.2.0 以降は、デーモンはAmazon ECRに掲載されています。イメージを引っ張る前にAmazon ECR パブリックレジストリにDocker クライアントを認証する必要があります。

次のコマンドを実行して、最新のリリース 3.x バージョンタグを引き出します。

```
docker pull public.ecr.aws/xray/aws-xray-daemon:3.x
```

以前のリリースまたはアルファ版は、3.xとalphaまたは特定のバージョン番号に置き換えてダウンロードできます。

本番環境では、アルファタグ付きのデーモンイメージを使用することはお勧めしません。

Docker Hub

デーモンは、Docker Hubで見ることができます。次のコマンドを実行して、最新リリースの3.xバージョンをダウンロードします。

```
docker pull amazon/aws-xray-daemon:3.x
```

デーモンの以前のリリースは、3.x希望のバージョンに置き換えてリリースすることができます。

デーモンアーカイブの署名の確認

GPG 署名ファイルは、ZIP アーカイブで圧縮されたデーモンアセットで使用するために含まれています。ホストのパブリックキーは、次の場所にあります。aws-xray.gpg

公開鍵を使用して、デーモンの ZIP アーカイブがオリジナルで変更されていないことを確認できます。まず、GnuPG で公開鍵をインポートします。

パブリックキーをインポートするには

1. 公開鍵をダウンロードします。

```
$ BUCKETURL=https://s3.us-east-2.amazonaws.com/aws-xray-assets.us-east-2
$ wget $BUCKETURL/xray-daemon/aws-xray.gpg
```

2. 公開鍵をキーリングにインポートします。

```
$ 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
```

インポートされたキーを使用してデーモンの ZIP アーカイブの署名を確認します。

アーカイブの署名を確認するには

1. アーカイブおよび署名ファイルをダウンロードします。

```
$ BUCKETURL=https://s3.us-east-2.amazonaws.com/aws-xray-assets.us-east-2
$ wget $BUCKETURL/xray-daemon/aws-xray-daemon-linux-3.x.zip
$ wget $BUCKETURL/xray-daemon/aws-xray-daemon-linux-3.x.zip.sig
```

2. `gpg --verify` を実行して署名を確認します。

```
$ 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
```

信頼性に関する警告に注意します。自分や信頼する人が署名した場合、鍵は信頼されます。これは、署名が無効であることを意味するものではなく、公開鍵を確認していないことを意味します。

デーモンを実行する

コマンドラインからローカルでデーモンを実行します。ローカルモードで実行するには `-o` オプションを、リージョンを設定するには `-n` を使用します。

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

プラットフォーム固有の詳細な手順については、以下のトピックを参照してください。

- Linux (ローカル) – Linux で X-Ray デーモンを実行する
- Windows (ローカル) – Windows で X-Ray デーモンを実行する
- Elastic Beanstalk – AWS Elastic Beanstalk で X-Ray デーモンを実行します
- Amazon EC2 – Amazon EC2 での X-Ray デーモンの実行
- Amazon ECS – Amazon ECS での X-Ray デーモンの実行

コマンドラインオプションまたは設定ファイルを使用して、デーモンの動作をさらにカスタマイズできます。詳細については、「AWS X-Ray デーモンの設定」を参照してください。

X-Rayにデータを送信するアクセス権限をデーモンに付与する

X-Ray デーモンは AWS SDK を使用してトレースデータを X-Ray にアップロードするため、そのためのアクセス許可を持つ認証情報が必要です AWS 。

Amazon EC2 では、デーモンはインスタンスのインスタンスプロファイルのロールを自動的に使用します。デーモンをローカルで実行するために必要な認証情報については、「アプリケーションをローカルで実行する」を参照してください。

1 つ以上の場所 (認証情報ファイル、インスタンスプロファイル、または環境変数) で認証情報を指定する場合、SDK プロバイダーチェーンは、使用される認証情報を決定します。SDK に認証情報を提供する方法の詳細については、 SDK for Go 開発者ガイドのAWS 認証情報の指定を参照してください。

デーモンの認証情報が属している IAM ロールまたはユーザーには、サービスにデータを書き込むアクセス権限が必要です。

- Amazon EC2 でデーモンを使用するには、新しいインスタンスプロファイルのロールを作成するか、既存のロールに管理ポリシーを追加します。
- Elastic Beanstalk でデーモンを使用するには、管理ポリシーを Elastic Beanstalk のデフォルトのインスタンスプロファイルのロールに追加します。
- デーモンをローカルで実行するには、「アプリケーションをローカルで実行する」を参照してください。

詳細については、「の Identity and Access Management AWS X-Ray」を参照してください。

X-Ray デーモンログ

デーモンは、現在の設定と に送信するセグメントに関する情報を出力します 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)
```

デフォルトでは、デーモンはログを STDOUT に出力します。デーモンをバックグラウンドで実行する場合は、`--log-file` コマンドラインオプションまたは設定ファイルを使用してログファイルパスを設定します。ログレベルを設定し、ログローテーションを無効にすることもできます。手順については、「AWS X-Ray デーモンの設定」を参照してください。

Elastic Beanstalk では、プラットフォームはデーモンログの場所を設定します。詳細については、「AWS Elastic Beanstalk で X-Ray デーモンを実行します」を参照してください。

AWS X-Ray デーモンの設定

コマンドラインオプションまたは設定ファイルを使用して、X-Ray デーモンの動作をカスタマイズできます。ほとんどのオプションは両方の方法を使用して利用できますが、一部は設定ファイルのみを使用でき、一部はコマンドラインのみを使用できます。

開始するには、`-n` または`--region`のみが必要です。デーモンがトレースデータをX-Ray に送信するために使用するリージョンを設定するために使用します。

```
~/xray-daemon$ ./xray -n us-east-2
```

デーモンを、Amazon EC2 ではなく、ローカルで実行している場合、`-o` オプションを追加すると、デーモンをより迅速に開始できるように、インスタンスプロファイルの認証情報のチェックをスキップできます。

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

残りのコマンドラインオプションを使用して、ロギングの設定、別のポートでのリッスン、デーモンが使用できるメモリ量の制限、別のアカウントにトレースデータを送信するロールの割り当てができます。

設定ファイルをデーモンに渡して、詳細な設定オプションにアクセスしたり、X-Ray への同時呼び出し数を制限したり、ログローテーションを無効にしたり、プロキシにトラフィックを送信したりすることができます。

セクション

- サポートされている環境変数

- コマンドラインオプションを使用する
- 設定ファイルを使用する

サポートされている環境変数

X-Ray デーモンは次の環境変数をサポートしています。

- AWS_REGION – X-Ray サービスエンドポイントの AWS リージョン を指定します。
- HTTPS_PROXY セグメントをアップロードするデーモンのプロキシアドレスを指定します。これは、DNS ドメイン名または IP アドレス、あるいはプロキシサーバーで使用されているポート番号のいずれかです。

コマンドラインオプションを使用する

ローカルで実行するか、またはユーザーデータスクリプトを使用して、これらのオプションをデーモンに渡します。

コマンドラインオプション

- -b、--bind 別の UDP ポートでセグメントドキュメントをリッスンします。

```
--bind "127.0.0.1:3000"
```

  デフォルト – 2000

- -t、 --bind-tcp 別の TCP ポートで X-Ray サービスへの呼び出しをリッスンします。

```
-bind-tcp "127.0.0.1:3000"
```

  デフォルト – 2000

- -c、 --config 指定されたパスから設定ファイルをロードします。

```
--config "/home/ec2-user/xray-daemon.yaml"
```

- -f、 --log-file 指定されたファイルパスにログを出力します。

```
--log-file "/var/log/xray-daemon.log"
```

- `-l`、 `--log-level`ログレベルを dev、debug、info、warn、error、prod (詳細な順) から指定します。

  ```
  --log-level warn
  ```

  デフォルト – prod
- `-m`、 `--buffer-memory`バッファが使用できるメモリの量をメガバイト単位で変更します (最小値 3)。

  ```
  --buffer-memory 50
  ```

  デフォルト – 使用可能なメモリ総量の 1%
- `-o`、 `--local-mode`– EC2 インスタンスのメタデータをチェックしません。
- `-r`、 `--role-arn`– 指定した IAM ロールで、別のアカウントにセグメントをアップロードできるようにします。

  ```
  --role-arn "arn:aws:iam::123456789012:role/xray-cross-account"
  ```
- `-a`、 `--resource-arn` — デーモンを実行するリソースの Amazon AWS リソースネーム (ARN)。
- `-p`、 `--proxy-address` — プロキシ AWS X-Ray 経由で にセグメントをアップロードします。プロキシサーバーのプロトコルを指定する必要があります。

  ```
  --proxy-address "http://192.0.2.0:3000"
  ```
- `-n`、`--region` 特定のリージョンの X-Ray サービスにセグメントを送信します。
- `-v`、 `--version` – AWS X-Ray デーモンのバージョンを表示します。
- `-h`、`--help` – ヘルプ画面を表示します。

設定ファイルを使用する

YAML 形式のファイルを使用して、デーモンを設定することもできます。`-c` オプションを使用して、設定ファイルをデーモンに渡します。

```
~$ ./xray -c ~/xray-daemon.yaml
```

設定ファイルのオプション

- TotalBufferSizeMB – 最大バッファサイズ (MB) (最小値 3)。ホストメモリの 1% を使用するには、0 を選択します。
- Concurrency - セグメントドキュメントをアップロード AWS X-Ray するための への同時呼び出しの最大数。
- Region – 特定のリージョンでセグメントを AWS X-Ray サービスに送信します。
- Socket – デーモンのバインディングを設定します。
  - UDPAddress – デーモンがリッスンするポートを変更します。
  - TCPAddress – 別の TCP ポートで X-Ray サービスへの呼び出しをリッスンします。
- Logging – ログ記録の動作を設定します。
  - LogRotation – falseに設定してログローテーションを無効にします。
  - LogLevel – ログレベルを最も詳細から最小に変更します: dev、debug、info、prodwarnerrorprod、。デフォルトは です。prodと等価である。info。
  - LogPath – 指定されたファイルパスにログを出力します。
- LocalMode–EC2 インスタンスのメタデータのチェックをスキップするには true に設定します。
- ResourceARN – デーモンを実行するリソースの Amazon AWS リソースネーム (ARN)。
- RoleARN – 指定した IAM ロールで、別のアカウントにセグメントをアップロードできるようにします。
- ProxyAddress — プロキシ AWS X-Ray 経由でセグメントを にアップロードします。
- Endpoint – デーモンがセグメントドキュメントを送信する X-Ray サービスエンドポイントを変更します。
- NoVerifySSL – TLS 証明書認証を無効にします。
- Version – デーモンの設定ファイル形式のバージョン。ファイル形式のバージョンは必須フィールド。

Example xray-daemon.yaml

この設定ファイルでは、デーモンのリスニングポートを 3000 に変更し、インスタンスメタデータの確認をオフにして、セグメントをアップロードするために使用するロールを設定し、リージョンおよびログ作成オプションを変更します。

```
Socket:
  UDPAddress: "127.0.0.1:3000"
  TCPAddress: "127.0.0.1:3000"
Region: "us-west-2"
Logging:
  LogLevel: "warn"
  LogPath: "/var/log/xray-daemon.log"
LocalMode: true
RoleARN: "arn:aws:iam::123456789012:role/xray-cross-account"
Version: 2
```

ローカルで X-Ray; デーモンを実行する

AWS X-Ray デーモンは、Linux、MacOS、Windows、または Docker コンテナでローカルに実行できます。実装されたアプリケーションを開発してテストするときに、デーモンを実行してトレースデータを X-Ray に中継します。ここに示す手順を使用して、デーモンをダウンロードして解凍します。

ローカルで実行する場合、デーモンは AWS SDK 認証情報ファイル (`.aws/credentials` ユーザーディレクトリの) または環境変数から認証情報を読み取ることができます。詳細については、「X-Rayにデータを送信するアクセス権限をデーモンに付与する」を参照してください。

デーモンはポート 2000 の UDP データをリッスンします。ポートおよびその他のオプションは、設定ファイルとコマンドラインオプションを使用して変更できます。詳細については、「AWS X-Rayデーモンの設定」を参照してください。

Linux で X-Ray デーモンを実行する

コマンドラインからデーモンの実行可能ファイルを実行できます。ローカルモードで実行するには `-o` オプションを、リージョンを設定するには `-n` を使用します。

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

バックグラウンドでデーモンを実行するには、`&` を使用します。

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

`pkill` を使用してバックグラウンドで実行されるデーモンプロセスを終了します。

```
~$ pkill xray
```

Docker コンテナで X-Ray デーモンを実行する

Docker コンテナでデーモンをローカルで実行するには、Dockerfile という名前のファイルに次のテキストを保存します。Amazon ECR で、完全なイメージ例をダウンロードします。詳細については、「デーモンのダウンロード」を参照してください。

Example Dockerfile — Amazon Linux

```
FROM amazonlinux
RUN yum install -y unzip
RUN curl -o daemon.zip https://s3.us-east-2.amazonaws.com/aws-xray-assets.us-east-2/
xray-daemon/aws-xray-daemon-linux-3.x.zip
RUN unzip daemon.zip && cp xray /usr/bin/xray
ENTRYPOINT ["/usr/bin/xray", "-t", "0.0.0.0:2000", "-b", "0.0.0.0:2000"]
EXPOSE 2000/udp
EXPOSE 2000/tcp
```

docker build を使用してコンテナイメージを構築します。

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

docker run を使用してコンテナでイメージを実行します。

```
~/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
```

このコマンドでは、以下のオプションを使用します。

- --attach STDOUT – ターミナルにデーモンからの出力を表示します。
- -v ~/.aws/:/root/.aws/:ro – コンテナに .aws ディレクトリへの読み取り専用アクセスを許可して、 AWS SDK 認証情報を読み取らせます。

- AWS_REGION=*us-east-2* – AWS_REGION環境変数を設定して、使用するリージョンをデーモンに指定します。
- --net=host – コンテナを host ネットワークにアタッチします。ホストネットワーク上のコンテナは、ポートを公開しなくても相互に通信できます。
- -p 2000:2000/udp – マシン上の UDP ポート 2000 をコンテナの同じポートにマップします。これは、同じネットワークのコンテナが通信するために必要ではありませんが、これにより、コマンドラインまたは Docker で実行されていないアプリケーションからセグメントをデーモンに送信できます。
- --name xray-daemon – ランダムな名前を生成する代わりに、コンテナに xray-daemon という名前を付けます。
- -o (イメージ名の後ろ) – コンテナ内でデーモンを実行するエントリポイントに -o オプションを追加します。このオプションは、Amazon EC2インスタンスのメタデータを読み取らないようにするため、ローカルモードで実行するようにデーモンに指示します。

デーモンを停止するには、docker stop を使用します。Dockerfile を変更して新しいイメージを作成する場合は、同じ名前の別のコンテナを作成する前に、既存のコンテナを削除する必要があります。docker rm を使用してコンテナを削除します。

```
$ docker stop xray-daemon
$ docker rm xray-daemon
```

Windows で X-Ray デーモンを実行する

コマンドラインからデーモンの実行可能ファイルを実行できます。ローカルモードで実行するには -o オプションを、リージョンを設定するには -n を使用します。

```
> .\xray_windows.exe -o -n us-east-2
```

PowerShell スクリプトを使用して、デーモンのサービスを作成して実行します。

Example PowerShell スクリプト - Windows

```
if ( Get-Service "AWSXRayDaemon" -ErrorAction SilentlyContinue ){
    sc.exe stop AWSXRayDaemon
    sc.exe delete AWSXRayDaemon
}
if ( Get-Item -path aws-xray-daemon -ErrorAction SilentlyContinue ) {
```

```
    Remove-Item -Recurse -Force aws-xray-daemon
}

$currentLocation = Get-Location
$zipFileName = "aws-xray-daemon-windows-service-3.x.zip"
$zipPath = "$currentLocation\$zipFileName"
$destPath = "$currentLocation\aws-xray-daemon"
$daemonPath = "$destPath\xray.exe"
$daemonLogPath = "C:\inetpub\wwwroot\xray-daemon.log"
$url = "https://s3.dualstack.us-west-2.amazonaws.com/aws-xray-assets.us-west-2/xray-
daemon/aws-xray-daemon-windows-service-3.x.zip"

Invoke-WebRequest -Uri $url -OutFile $zipPath
Add-Type -Assembly "System.IO.Compression.Filesystem"
[io.compression.zipfile]::ExtractToDirectory($zipPath, $destPath)

sc.exe create AWSXRayDaemon binPath= "$daemonPath -f $daemonLogPath"
sc.exe start AWSXRayDaemon
```

OS X で X-Ray デーモンを実行する

コマンドラインからデーモンの実行可能ファイルを実行できます。ローカルモードで実行するには -o オプションを、リージョンを設定するには -n を使用します。

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

バックグラウンドでデーモンを実行するには、& を使用します。

```
~/xray-daemon$ ./xray_mac -o -n us-east-2 &
```

ターミナル終了時にデーモンが終了されるのを防止するには、nohup を使用します。

```
~/xray-daemon$ nohup ./xray_mac &
```

AWS Elastic Beanstalk で X-Ray デーモンを実行します

アプリケーションからのトレースデータを AWS X-Ray に中継するには、Elastic Beanstalk 環境の Amazon EC2 インスタンスで X-Ray デーモンを実行します。サポートされているプラットフォームの一覧は、[AWS Elastic Beanstalk Developer Guide] (開発者ガイド) の [Configuring AWS X-Ray Debugging] (デバッグ設定)を参照してください。

> **Note**
>
> デーモンは、環境のインスタンスプロファイルのアクセス権限を使用します。Elastic Beanstalk インスタンスプロファイルにアクセス権限を追加する方法については、X-Rayにデータを送信するアクセス権限をデーモンに付与する を参照してください。

Elastic Beanstalk プラットフォームには、デーモンを自動的に実行する設定オプションがあります。ソースコードの設定ファイル内で、または Elastic Beanstalk コンソールでオプションを選択して、デーモンを有効にできます。設定オプションを有効にすると、デーモンはインスタンスにインストールされ、サービスとして実行されます。

Elastic Beanstalk プラットフォームに含まれているバージョンは最新バージョンではない場合があります。サポートされているプラットフォームのトピックを参照して、プラットフォーム設定で利用できるデーモンのバージョンを確認してください。

Elastic Beanstalk は、複数コンテナの Docker (Amazon ECS) プラットフォームに X-Ray デーモンを提供しません。

Elastic Beanstalk X-Ray 統合を使用して X-Ray デーモンの実行します。

コンソールを使用して X-Ray 統合をオンにするか、設定ファイルを使用してアプリケーションソースコードで設定します。

Elastic Beanstalk コンソールで X-Ray デーモンを有効にするには

1. [Elastic Beanstalk console] (Elastic Beanstalk コンソール) を開いてください。
2. 環境に対応するマネジメントコンソールに移動します。
3. [設定] を選択します。
4. [Software Settings] を選択します。
5. [X-Ray daemon] で、[Enabled] を選択します。
6. [Apply (適用)] を選択します。

ソースコードに設定ファイルを含めて、設定を環境間で移植可能にできます。

Example .ebextensions/xray-daemon.config

```
option_settings:
```

```
  aws:elasticbeanstalk:xray:
    XRayEnabled: true
```

Elastic Beanstalkは設定ファイルをデーモンに渡し、ログを標準の場所に出力します。

Windows Server プラットフォーム

- [Configuration file] (設定ファイル) – `C:\Program Files\Amazon\XRay\cfg.yaml`
- [Logs] (ログ) – `c:\Program Files\Amazon\XRay\logs\xray-service.log`

Linux プラットフォーム

- [Configuration file] (設定ファイル) – `/etc/amazon/xray/cfg.yaml`
- [Logs] (ログ) – `/var/log/xray/xray.log`

Elastic Beanstalkには、AWS Management Console またはコマンドラインからインスタンスログを取得するためのツールが用意されています。設定ファイルでタスクを追加して、X-Ray デーモンログを含めるように Elastic Beanstalk に伝えることができます。

Example .ebextensions/xray-logs.config - Linux

```
files:
  "/opt/elasticbeanstalk/tasks/taillogs.d/xray-daemon.conf" :
    mode: "000644"
    owner: root
    group: root
    content: |
      /var/log/xray/xray.log
```

Example .ebextensions/xray-logs.config - Windows Server

```
files:
  "c:/Program Files/Amazon/ElasticBeanstalk/config/taillogs.d/xray-daemon.conf" :
    mode: "000644"
    owner: root
    group: root
    content: |
      c:\Progam Files\Amazon\XRay\logs\xray-service.log
```

詳細については、[AWS Elastic Beanstalk Developer Guide] (開発者ガイド)の[Viewing Logs from Your Elastic Beanstalk Environment's Amazon EC2 Instances] (Elastic Beanstalk 環境のAmazon EC2 インスタンスからのログの表示) を参照してください。

X-Ray デーモンを手動でダウンロードして実行します (上級編)

X-Ray デーモンをプラットフォーム設定で使用できない場合は、Amazon S3 からダウンロードして、設定ファイルを使用して実行できます。

Elastic Beanstalk 設定 ファイルを使用して、デーモンをダウンロードして実行します。

Example .ebextensions/xray.config - Linux

```
commands:
  01-stop-tracing:
    command: yum remove -y xray
    ignoreErrors: true
  02-copy-tracing:
    command: curl https://s3.us-east-2.amazonaws.com/aws-xray-assets.us-east-2/xray-
daemon/aws-xray-daemon-3.x.rpm -o /home/ec2-user/xray.rpm
  03-start-tracing:
    command: yum install -y /home/ec2-user/xray.rpm

files:
  "/opt/elasticbeanstalk/tasks/taillogs.d/xray-daemon.conf" :
    mode: "000644"
    owner: root
    group: root
    content: |
      /var/log/xray/xray.log
  "/etc/amazon/xray/cfg.yaml" :
    mode: "000644"
    owner: root
    group: root
    content: |
      Logging:
        LogLevel: "debug"
      Version: 2
```

Example .ebextensions/xray.config - Windows server

```
container_commands:
  01-execute-config-script:
```

```
    command: Powershell.exe -ExecutionPolicy Bypass -File c:\\temp\\installDaemon.ps1
    waitAfterCompletion: 0

files:
  "c:/temp/installDaemon.ps1":
    content: |
      if ( Get-Service "AWSXRayDaemon" -ErrorAction SilentlyContinue ) {
          sc.exe stop AWSXRayDaemon
          sc.exe delete AWSXRayDaemon
      }

      $targetLocation = "C:\Program Files\Amazon\XRay"
      if ((Test-Path $targetLocation) -eq 0) {
          mkdir $targetLocation
      }

      $zipFileName = "aws-xray-daemon-windows-service-3.x.zip"
      $zipPath = "$targetLocation\$zipFileName"
      $destPath = "$targetLocation\aws-xray-daemon"
      if ((Test-Path $destPath) -eq 1) {
          Remove-Item -Recurse -Force $destPath
      }

      $daemonPath = "$destPath\xray.exe"
      $daemonLogPath = "$targetLocation\xray-daemon.log"
      $url = "https://s3.dualstack.us-west-2.amazonaws.com/aws-xray-assets.us-west-2/
xray-daemon/aws-xray-daemon-windows-service-3.x.zip"

      Invoke-WebRequest -Uri $url -OutFile $zipPath
      Add-Type -Assembly "System.IO.Compression.Filesystem"
      [io.compression.zipfile]::ExtractToDirectory($zipPath, $destPath)

      New-Service -Name "AWSXRayDaemon" -StartupType Automatic -BinaryPathName
 "`"$daemonPath`" -f `"$daemonLogPath`""
      sc.exe start AWSXRayDaemon
    encoding: plain
  "c:/Program Files/Amazon/ElasticBeanstalk/config/taillogs.d/xray-daemon.conf" :
    mode: "000644"
    owner: root
    group: root
    content: |
      C:\Program Files\Amazon\XRay\xray-daemon.log
```

これらの例ではまた、デーモンのログファイルを のログ末尾タスクに追加し、コンソールまたは Elastic Beanstalk コマンドラインインターフェイス (EB CLI) を使用したログのリクエスト時にその内容を含めるようにしています。

Amazon EC2 での X-Ray デーモンの実行

Amazon EC2 上の次のオペレーティングシステムで X-Ray デーモンを実行できます。

- Amazon Linux
- Ubuntu
- Windows Server (2012 R2 以降)

インスタンスプロファイルを使用して、トレースデータを X-Ray にアップロードするアクセス権限をデーモンに付与します。詳細については、「X-Rayにデータを送信するアクセス権限をデーモンに付与する」を参照してください。

インスタンスを起動するときに、ユーザーデータのスクリプトを使用して自動的にデーモンを実行します。

Example ユーザーデータスクリプト - Linux

```
#!/bin/bash
curl https://s3.us-east-2.amazonaws.com/aws-xray-assets.us-east-2/xray-daemon/aws-xray-daemon-3.x.rpm -o /home/ec2-user/xray.rpm
yum install -y /home/ec2-user/xray.rpm
```

Example ユーザーデータスクリプト - Windows Server

```
<powershell>
if ( Get-Service "AWSXRayDaemon" -ErrorAction SilentlyContinue ) {
    sc.exe stop AWSXRayDaemon
    sc.exe delete AWSXRayDaemon
}

$targetLocation = "C:\Program Files\Amazon\XRay"
if ((Test-Path $targetLocation) -eq 0) {
    mkdir $targetLocation
}
```

```
$zipFileName = "aws-xray-daemon-windows-service-3.x.zip"
$zipPath = "$targetLocation\$zipFileName"
$destPath = "$targetLocation\aws-xray-daemon"
if ((Test-Path $destPath) -eq 1) {
    Remove-Item -Recurse -Force $destPath
}

$daemonPath = "$destPath\xray.exe"
$daemonLogPath = "$targetLocation\xray-daemon.log"
$url = "https://s3.dualstack.us-west-2.amazonaws.com/aws-xray-assets.us-west-2/xray-
daemon/aws-xray-daemon-windows-service-3.x.zip"

Invoke-WebRequest -Uri $url -OutFile $zipPath
Add-Type -Assembly "System.IO.Compression.Filesystem"
[io.compression.zipfile]::ExtractToDirectory($zipPath, $destPath)

New-Service -Name "AWSXRayDaemon" -StartupType Automatic -BinaryPathName
 "`"$daemonPath`" -f `"$daemonLogPath`""
sc.exe start AWSXRayDaemon
</powershell>
```

Amazon ECS での X-Ray デーモンの実行

Amazon ECS で、X-Ray デーモンを実行する Docker イメージを作成し、それを Docker イメージリポジトリにアップロードして、 クラスターにデプロイできます。タスク定義ファイルでポートマッピングとネットワークモード設定を使用すると、アプリケーションがデーモンコンテナと通信できるようになります。

公式 Docker イメージの使用

X-Ray は、アプリケーションと一緒にデプロイできる Amazon ECRのDocker コンテナイメージを提供します。詳細については、「デーモンのダウンロード」を参照してください。

Example タスク定義

```
    {
      "name": "xray-daemon",
      "image": "amazon/aws-xray-daemon",
      "cpu": 32,
      "memoryReservation": 256,
```

```
      "portMappings" : [
          {
              "hostPort": 0,
              "containerPort": 2000,
              "protocol": "udp"
          }
       ]
    }
```

Docker イメージの作成と構築

カスタム設定では、独自の Docker イメージの定義が必要になる場合があります。

タスクロールに管理ポリシーを追加して、デーモンにトレースデータを X-Ray にアップロードするアクセス許可を与えます。詳細については、「X-Rayにデータを送信するアクセス権限をデーモンに付与する」を参照してください。

次のいずれかの Dockerfiles を使用して、デーモンを実行するイメージを作成します。

Example Dockerfile — Amazon Linux

```
FROM amazonlinux
RUN yum install -y unzip
RUN curl -o daemon.zip https://s3.us-east-2.amazonaws.com/aws-xray-assets.us-east-2/
xray-daemon/aws-xray-daemon-linux-3.x.zip
RUN unzip daemon.zip && cp xray /usr/bin/xray
ENTRYPOINT ["/usr/bin/xray", "-t", "0.0.0.0:2000", "-b", "0.0.0.0:2000"]
EXPOSE 2000/udp
EXPOSE 2000/tcp
```

> **Note**
>
> マルチコンテナ環境のループバックをリッスンするためのバインディングアドレスを指定するには、フラグ `-t` および `-b` が必要です。

Example Dockerfile - Ubuntu

Debian から派生した OS では、認証機関 (CA) の証明書をインストールして、インストーラのダウンロード時の問題を回避します。

```
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
```

タスク定義では、使用するネットワーキングモードによって設定が異なります。デフォルトはブリッジネットワーキングで、デフォルトの VPC で使用できます。ブリッジネットワークで、X-Ray SDK に参照先のコンテナポートを指示し、ホストポートを設定するように AWS_XRAY_DAEMON_ADDRESS 環境変数を設定します。たとえば、UDP ポート 2000 を発行し、アプリケーションコンテナからデーモンコンテナへのリンクを作成します。

Example タスク定義

```
    {
      "name": "xray-daemon",
      "image": "123456789012.dkr.ecr.us-east-2.amazonaws.com/xray-daemon",
      "cpu": 32,
      "memoryReservation": 256,
      "portMappings" : [
          {
              "hostPort": 0,
              "containerPort": 2000,
              "protocol": "udp"
          }
       ]
    },
    {
      "name": "scorekeep-api",
      "image": "123456789012.dkr.ecr.us-east-2.amazonaws.com/scorekeep-api",
      "cpu": 192,
      "memoryReservation": 512,
      "environment": [
          { "name" : "AWS_REGION", "value" : "us-east-2" },
          { "name" : "NOTIFICATION_TOPIC", "value" : "arn:aws:sns:us-
east-2:123456789012:scorekeep-notifications" },
```

```
            { "name" : "AWS_XRAY_DAEMON_ADDRESS", "value" : "xray-daemon:2000" }
        ],
        "portMappings" : [
            {
                "hostPort": 5000,
                "containerPort": 5000
            }
        ],
        "links": [
          "xray-daemon"
        ]
    }
```

VPC のプライベートサブネットでクラスターを実行する場合は、awsvpc ネットワークモードを使用して、Elastic Network Interface (ENI) をコンテナにアタッチできます。これにより、リンクの使用を避けることができます。ポートマッピング、リンク、および AWS_XRAY_DAEMON_ADDRESS 環境変数でホストポートを省略します。

Example VPC タスク定義

```
{
    "family": "scorekeep",
    "networkMode":"awsvpc",
    "containerDefinitions": [
        {
          "name": "xray-daemon",
          "image": "123456789012.dkr.ecr.us-east-2.amazonaws.com/xray-daemon",
          "cpu": 32,
          "memoryReservation": 256,
          "portMappings" : [
              {
                  "containerPort": 2000,
                  "protocol": "udp"
              }
          ]
        },
        {
            "name": "scorekeep-api",
            "image": "123456789012.dkr.ecr.us-east-2.amazonaws.com/scorekeep-api",
            "cpu": 192,
            "memoryReservation": 512,
            "environment": [
                { "name" : "AWS_REGION", "value" : "us-east-2" },
```

```
                { "name" : "NOTIFICATION_TOPIC", "value" : "arn:aws:sns:us-
east-2:123456789012:scorekeep-notifications" }
            ],
            "portMappings" : [
                {
                    "containerPort": 5000
                }
            ]
        }
    ]
}
```

Amazon ECS コンソールでのコマンドラインオプションの構成

コマンドラインオプションは、イメージの設定ファイル内の競合する値を上書きします。コマンドラインオプションは、一般的にローカルテストに使用されますが、環境変数を設定する際の便宜上、または起動プロセスを制御するために使用することもできます。

コマンドラインオプションを追加することで、コンテナに渡される Docker CMD が更新されます。詳細については、Docker run リファレンスを参照してください。

コマンドラインオプションを設定するには

1. Amazon ECS クラシックコンソール (https://console.aws.amazon.com/ecs/) を開きます。
2. ナビゲーションバーから、タスク定義を含むリージョンを選択します。
3. ナビゲーションペインで、[タスク定義] を選択します。
4. [Task Definitions] ページで、変更するタスク定義の左側にあるボックスをオンにし、[Create new revision] を選択します。
5. [Create new revision of Task Definition (タスク定義の新しいリビジョンを作成)] ページで、コンテナを選択します。
6. [ENVIRONMENT (環境)] セクションで、コマンドラインオプションのカンマ区切りリストを [Command (コマンド)] フィールドに追加します。
7. [更新] を選択します。
8. 情報を確認し、[Create] を選択します。

次の例は、RoleARN オプションのコマンドラインオプションをカンマで区切って記述する方法を示しています。RoleARN オプションは、セグメントを別のアカウントにアップロードするために、指定された IAM ロール を引き受けます。

Example

```
--role-arn, arn:aws:iam::123456789012:role/xray-cross-account
```

X-Ray で使用可能なコマンドラインオプションの詳細については、「AWS X-Rayデーモンの設定」を参照してください。

のアプリケーションを計測する AWS X-Ray

アプリケーションを計測するには、アプリケーション内の受信と送信リクエストおよびその他のイベントのトレースデータを、各リクエストに関するメタデータと一緒に送信する必要があります。特定の要件に基づいて、選択または組み合わせることができる計測オプションがいくつかあります。

- 自動計測 – 通常、設定の変更または自動計測エージェントや他のメカニズムを追加して、コードをゼロに変更してアプリケーションを計測します。
- ライブラリインストルメンテーション — アプリケーションコードの変更を最小限に抑え、 AWS SDK、Apache HTTP クライアント、SQL クライアントなどの特定のライブラリやフレームワークを対象とする構築済みのインストルメンテーションを追加します。
- 手動計測 – トレース情報を送信する各場所で、アプリケーションに計測コードを追加します。

X-Ray トレース用のアプリケーションの計測に使用できる SDK、エージェント、およびツールがいくつかあります。

トピック

AWS Distro for を使用したアプリケーションの計測 OpenTelemetry

AWS Distro for OpenTelemetry (ADOT) は、Cloud Native Computing Foundation (CNCF) OpenTelemetry project. OpenTelemetry project に基づく AWS ディストリビューションです。 は、

分散トレースとメトリクスを収集するためのオープンソース APIs 、ライブラリ、およびエージェントの 1 セットを提供します。このツールキットは、 SDKs自動計測エージェント、コレクターなどのアップストリーム OpenTelemetry コンポーネントのディストリビューションであり、 によってテスト、最適化、保護、サポートされています AWS。

ADOT を使用すると、エンジニアはアプリケーションを一度計測し、相関メトリクスとトレースを Amazon CloudWatch、 AWS X-Ray、Amazon OpenSearch Service などの複数の AWS モニタリングソリューションに送信できます。

ADOT で X-Ray を使用するには、X-Ray での使用が有効になっている OpenTelemetry SDK と X-Ray での使用が有効になっている AWS Distro for OpenTelemetry Collector の 2 つのコンポーネントが必要です。 AWS X-Ray およびその他の OpenTelemetry で AWS Distro for を使用する方法の詳細については AWS のサービス、 AWS Distro for OpenTelemetry Documentation を参照してください。

言語のサポートと使用方法の詳細については、AWS 「 のオブザーバビリティ GitHub」を参照してください。

> **Note**
>
> CloudWatch エージェントを使用して、Amazon EC2 インスタンスとオンプレミスサーバーからメトリクス、ログ、トレースを収集できるようになりました。 CloudWatch エージェントバージョン 1.300025.0 以降では、 OpenTelemetryまたは X-Ray クライアント SDKs からトレースを収集して X-Ray に送信できます。Distro for OpenTelemetry (ADOT) Collector AWS または X-Ray デーモンの代わりに CloudWatch エージェントを使用してトレースを収集することで、管理するエージェントの数を減らすことができます。詳細については、「 CloudWatch ユーザーガイド」のCloudWatch 「 エージェント」トピックを参照してください。

ADOT には以下が含まれます。

- AWS Distro for OpenTelemetry Go
- AWS Distro for OpenTelemetry Java
- AWS Distro for OpenTelemetry JavaScript
- AWS Distro for OpenTelemetry Python
- AWS Distro for OpenTelemetry .NET

ADOT は現在、Javaおよ びPython用の自動計測サポートを含んでいます。さらに、ADOT は ADOT Managed AWS Lambda Layers を介して、Java、Node.js、Python ランタイムを使用した Lambda 関数とそのダウンストリームリクエストの自動計測を有効にします。 https://aws-otel.github.io/docs/getting-started/lambda

Java および Go 用の ADOT SDK は、X-Ray の一元化されたサンプリングルールをサポートしています。他の言語で X-Ray サンプリングルールのサポートが必要な場合は、 AWS X-Ray SDK の使用を検討してください。

> **Note**
>
> W3C トレース ID を X-Ray に送信できるようになりました。デフォルトでは、 で作成されたトレース OpenTelemetry には、W3C トレースコンテキスト仕様 に基づくトレース ID 形式があります。これは、X-Ray SDK を使用して作成されたトレース IDsの形式、または X-Ray と統合されている のサービスによって異なります AWS 。W3C 形式のトレース IDs が X-Ray で確実に受け入れられるようにするには、ADOT Collector バージョン 0.34.0 以降に含まれている AWS X-Ray Exporter バージョン 0.86.0 以降を使用する必要があります。以前のバージョンのエクスポーターはトレース ID のタイムスタンプを検証し、W3C トレース IDs が拒否される可能性があります。

AWS X-Ray SDKでアプリケーションを計測する

AWS X-Ray には、X-Ray にトレースを送信するためにアプリケーションを計測するための言語固有の SDKs のセットが含まれています。各 X-Ray SDK は、以下を提供します。

- インターセプター コードに追加して受信 HTTP リクエストをトレースする
- アプリケーションが他の を呼び出すために使用する AWS SDK クライアントを計測するクライアントハンドラー AWS のサービス
- HTTP クライアント 他の内部および外部 HTTP ウェブサービス呼び出しを計測する

X-Ray SDKsSQL データベースへの呼び出しの計測、 AWS SDK クライアントの自動計測、およびその他の機能もサポートしています。トレースデータを直接 X-Ray に送信する代わりに、SDK は JSON セグメントドキュメントを UDP トラフィックをリッスンしているデーモンプロセスに送信します。X-Ray デーモンはセグメントをキューにバッファし、バッチで X-Ray にアップロードします。

次の言語別の SDK が用意されています。

- AWS X-Ray SDK for Go
- AWS X-Ray SDK for Java
- AWS X-Ray SDK for Node.js
- AWS X-Ray SDK for Python
- AWS X-Ray SDK for .NET
- AWS X-Ray SDK for Ruby

X-Ray は現在、Java用の自動計測サポートを含んでいます。

AWS Distro for OpenTelemetry と X-Ray SDKs の選択

X-Ray に含まれる SDK は、 AWSによって提供される緊密に統合された計測ソリューションの一部です。 AWS Distro for OpenTelemetry は、X-Ray が多くのトレースソリューションの 1 つにすぎない、より広範な業界ソリューションの一部です。どちらの方法でも X-Ray で end-to-end トレースを実装できますが、最も有用なアプローチを決定するには、違いを理解することが重要です。

以下 OpenTelemetry が必要な場合は、 用 AWS Distro を使用してアプリケーションを計測することをお勧めします。

- コードを再計測することなく、複数の異なるトレースバックエンドにトレースを送信する機能
- OpenTelemetry コミュニティによって維持される、言語ごとに多数のライブラリインストルメンテーションのサポート
- Java、Python、Node.js を使用するときにコード変更する必要がない、テレメトリデータの収集に必要なすべてがパッケージ化された完全マネージド型の Lambda レイヤー

> **Note**
>
> AWS Distro for OpenTelemetry は、Lambda 関数の計測をより簡単に開始できます。ただし、柔軟性 OpenTelemetry により、Lambda 関数に追加のメモリが必要になり、呼び出しでコールドスタートのレイテンシーが増加し、追加料金が発生する可能性があります。低レイテンシー用に最適化していて、バックエンドの送信先を動的に設定できるOpenTelemetryの高度な機能を必要としない場合は、 AWS X-Ray SDK を使用してアプリケーションを計測できます。

以下が必要な場合は、アプリケーションの計測に X-Ray SDK を選択することをお勧めします。

- 緊密に統合されたシングルベンダーソリューション
- Node.js、Python、Ruby、.NET を使用する場合に、X-Ray コンソールからサンプリングルールを設定し、複数のホスト間で自動的に使用する機能を含む、X-Ray の一元化されたサンプリングルールとの統合

を使用してアプリケーションを計測する Go

X-Ray にトレースを送信するようにGoアプリケーションを計測するには、次の 2 つの方法があります。

- AWS Distro for OpenTelemetry Go – AWS Distro for OpenTelemetry Collector を介して相関メトリクスとトレースを Amazon CloudWatch、 AWS X-Ray、Amazon OpenSearch Service などの複数の AWS モニタリングソリューションに送信するための一連のオープンソースライブラリを提供するディス AWS トリビューション。
- AWS X-Ray SDK for Go – X-Ray デーモン を介してトレースを生成して X-Ray に送信するためのライブラリのセット。

詳細については、「AWS Distro for OpenTelemetry と X-Ray SDKs の選択」を参照してください。

AWS Distro for OpenTelemetry Go の使用

AWS Distro for OpenTelemetry Go を使用すると、アプリケーションを一度計測し、相関のあるメトリクスとトレースを Amazon CloudWatch、AWS X-Ray、Amazon OpenSearch Service などの複数の AWS モニタリングソリューションに送信することができます。X-Ray の使用AWSOpenElementry のディストリビューションには 2 つのコンポーネントが必要です。OpenTelemetry SDKX-Ray での使用が可能であり、AWSDistro for OpenTelemetry コレクターX-Ray での使用が有効。

開始するには、「」を参照してください。AWSDistro for OpenTelemetry Go ドキュメント。

AWS Distro for OpenTelemetry と AWS X-Ray やその他の AWS のサービス の併用については、「AWS Distro for OpenTelemetry」または「AWS Distro for OpenTelemetry Documentation」を参照してください。

言語サポートと使用方法の詳細については、「AWS Observability on Github」を参照してください。

AWS X-Ray SDK for Go

は、X-Ray SDK for Go アプリケーション用のライブラリのセットです。トレースデータを作成して X-Ray デーモンに送信するためのクラスとメソッドを提供します。トレースデータには、アプリケーションによって処理された受信 HTTP リクエスト、およびアプリケーションから AWS SDK を使用したダウンストリームサービス、HTTP クライアント、または SQL データベースコネクターに対して行われる呼び出しに関する情報が含まれています。セグメントを手動で作成し、注釈およびメタデータにデバッグ情報を追加することもできます。

`go get` を使用して GitHub リポジトリから SDK をダウンロードします。

```
$ go get -u github.com/aws/aws-xray-sdk-go/...
```

ウェブアプリケーションの場合は、`xray.Handler` 関数を使用して受信リクエストをトレースします。メッセージハンドラーでは、トレース対象リクエストごとに「セグメント」を作成し、レスポンスが送信されるとセグメントを完了します。セグメントが開いている間、SDK クライアントのメソッドを使用してセグメントに情報を追加し、サブセグメントを作成してダウンストリーム呼び出しをトレースできます。また、SDK では、セグメントが開いている間にアプリケーションがスローする例外を自動的に記録します。

インストルメント済みアプリケーションまたはサービスによって呼び出される Lambda 関数の場合、Lambda は トレースヘッダー を読み取り、サンプリングされたリクエストを自動的にトレースします。その他の関数については、Lambda の設定 から受信リクエストのサンプリングとトレースを行うことができます。いずれの場合も、Lambda はセグメントを作成し、X-Ray SDK に提供します。

> **Note**
>
> Lambda では、X-Ray SDK はオプションです。関数でこれを使用しない場合、サービスマップには Lambda サービスのノードと Lambda 関数ごとに 1 つのノードが含まれます。SDK を追加することで、関数コードをインストルメントして、Lambda で記録された関数セグメントにサブセグメントを追加することができます。詳細については、「AWS Lambda および AWS X-Ray」を参照してください。

次に、クライアントをAWS 関数の呼び出しでラップします。このステップでは、X-Ray 計測が任意のクライアントメソッドを呼び出すようにします。 計測が SQL データベースを呼び出すようにすることもできます。

SDK を入手したら、レコーダーとミドルウェアを設定して、その動作をカスタマイズします。プラグインを追加して、アプリケーションを実行しているコンピューティングリソースに関するデータを記録したり、サンプリングルールを定義することでサンプリングの動作のカスタマイズしたり、アプリケーションログに SDK からの情報をより多くあるいは少なく表示するようにログレベルを設定できます。

アプリケーションが注釈やメタデータで行うリクエストや作業に関する追加情報を記録します。注釈は、フィルタ式で使用するためにインデックス化されたシンプルなキーと値のペアで、特定のデータが含まれているトレースを検索できます。メタデータのエントリは制約が緩やかで、JSON にシリアル化できるオブジェクトと配列全体を記録できます。

> **注釈とメタデータ**
>
> 注釈およびメタデータとは、X-Ray SDK を使用してセグメントに追加する任意のテキストです。注釈は、フィルタ式用にインデックス付けされます。メタデータはインデックス化されませんが、X-Ray コンソールまたは API を使用して raw セグメントで表示できます。X-Ray への読み取りアクセスを許可した人は誰でも、このデータを表示できます。

コードに多数の計測されたクライアントがある場合、単一のリクエストセグメントには計測されたクライアントで行われた呼び出しごとに 1 個の多数のサブセグメントを含めることができます。カスタムサブセグメントで、クライアント呼び出しをラップすることで、サブセグメントを整理してグループできます。関数全体またはコードの任意のセクションのサブセグメントを作成し、親セグメントにすべてのレコードを記述する代わりにサブセグメントにメタデータと注釈を記録できます。

要件

X-Ray SDK for Go 1.9 以降が必要です。

SDK は、コンパイル時および実行時に次のライブラリに依存します。

- AWS SDK for Go バージョン 1.10.0 以降

これらの依存関係は SDK の `README.md` ファイルで宣言されています。

リファレンスドキュメント

SDK をダウンロードしたら、ドキュメントをビルドしてローカルにホストし、ウェブブラウザで表示します。

リファレンスドキュメントを表示するには

1. $GOPATH/src/github.com/aws/aws-xray-sdk-go (Linux または Mac) ディレクトリまたは %GOPATH%\src\github.com\aws\aws-xray-sdk-go (Windows) フォルダに移動します。
2. godoc コマンドを実行します。

```
$ godoc -http=:6060
```

3. http://localhost:6060/pkg/github.com/aws/aws-xray-sdk-go/ でブラウザを開きます。

X-Ray SDK for Goの設定

環境変数を使って X-Ray SDK for Go の設定を指定するには、Config オブジェクトで Configure を呼び出すか、デフォルト値を使用します。環境変数は Config 値よりも優先されます。これはデフォルト値よりも優先されます。

セクション

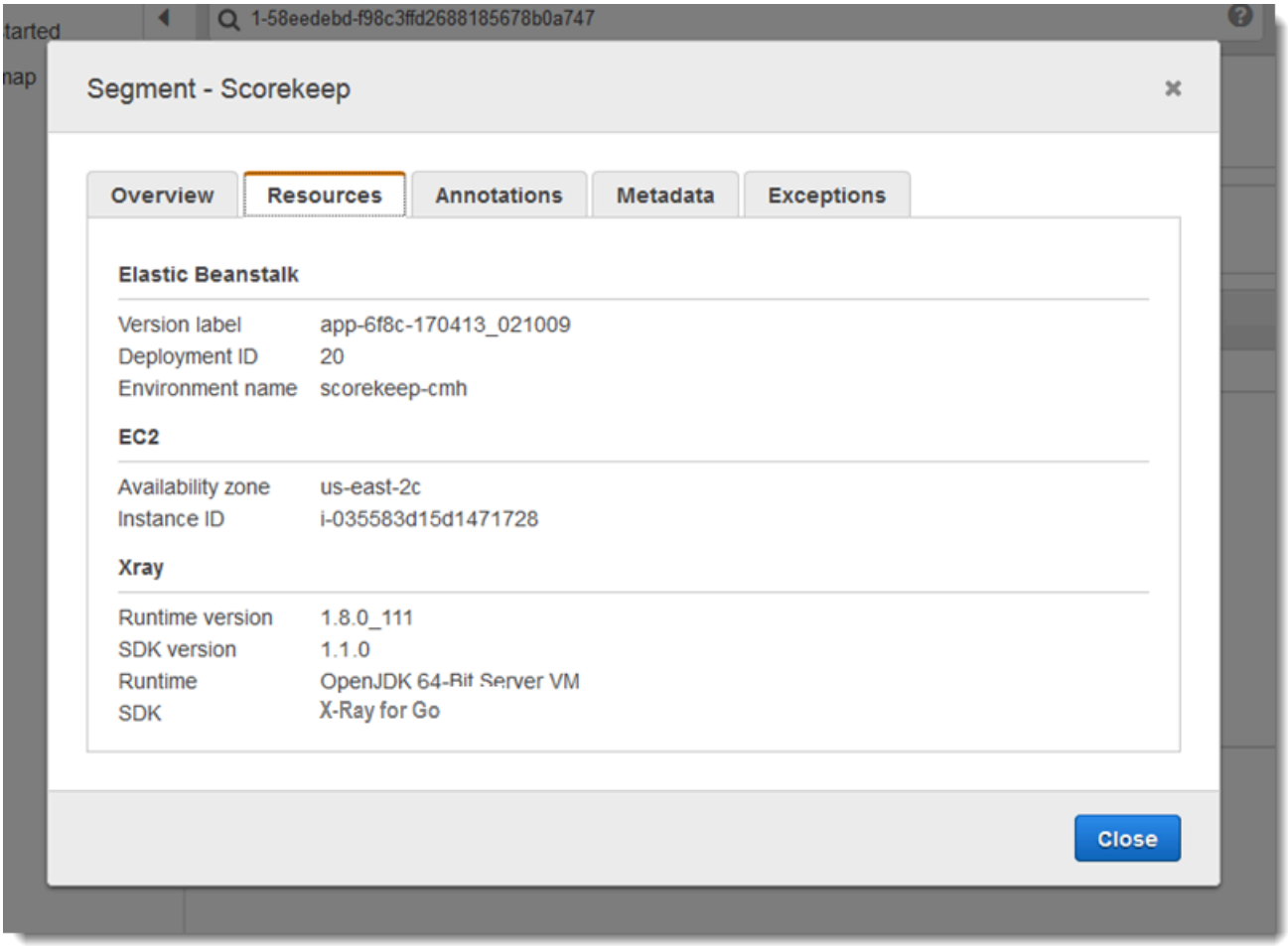
- サービスプラグイン
- サンプリングルール
- ログ記録
- 環境変数
- 設定の使用

サービスプラグイン

pluginsを使用して、アプリケーションをホストしているサービスに関する情報を記録します。

プラグイン

- Amazon EC2 – は、インスタンス ID、アベイラビリティーゾーン、および CloudWatch Logs グループEC2Pluginを追加します。
- ElasticBeanstalk– ElasticBeanstalkPluginは、環境名、バージョンラベル、およびデプロイ ID を追加します。
- Amazon ECS —ECSPluginは、コンテナ ID を追加します。



プラグインを使用するには、次のいずれかのパッケージをインポートします。

```
"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"
```

各プラグインには、プラグインをロードする明示的な `Init()` 関数呼び出しがあります。

Example ec2.Init()

```
import (
 "os"

 "github.com/aws/aws-xray-sdk-go/awsplugins/ec2"
 "github.com/aws/aws-xray-sdk-go/xray"
)

func init() {
```

```
  // conditionally load plugin
  if os.Getenv("ENVIRONMENT") == "production" {
    ec2.Init()
  }

  xray.Configure(xray.Config{
    ServiceVersion: "1.2.3",
  })
}
```

SDK はプラグイン設定を使用して、originセグメントのフィールド。これは、アプリケーションを実行する AWS リソースのタイプを示します。複数のプラグインを使用する場合、SDK は次の解決順序を使用してオリジンを決定します： ElasticBeanstalk > EKS > ECS > EC2。

サンプリングルール

SDK は X-Ray コンソールで定義したサンプリングルールを使用し、記録するリクエストを決定します。デフォルトルールでは、最初のリクエストを毎秒トレースし、X-Ray にトレースを送信するすべてのサービスで追加のリクエストの 5% をトレースします。X-Ray コンソールに追加のルールを作成するをクリックして、各アプリケーションで記録されるデータ量をカスタマイズします。

SDK は、定義された順序でカスタムルールを適用します。リクエストが複数のカスタムルールと一致する場合、SDK は最初のルールのみを適用します。

> **Note**
>
> SDK がサンプリングルールを取得するために X-Ray に到達できない場合、1 秒ごとに受信された最初のリクエストのデフォルトのローカルルールに戻り、ホストあたりの追加リクエストの 5% に戻ります。これは、ホストがサンプリング API を呼び出す権限を持っていない場合や、SDK によって行われる API 呼び出しの TCP プロキシとして機能する X-Ray デーモンに接続できない場合に発生します。

JSON ドキュメントからサンプリングルールをロードするように SDK を設定することもできます。SDK は、X-Ray サンプリングが利用できない場合のバックアップとしてローカルルールを使用することも、ローカルルールを排他的に使用することもできます。

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
  }
}
```

この例では、1 つのカスタムルールとデフォルトルールを定義します。カスタムルールでは、5 パーセントのサンプリングレートが適用され、/api/move/以下のパスに対してトレースするリクエストの最小数はありません。デフォルトのルールでは、1秒ごとの最初のリクエストおよび追加リクエストの 10 パーセントをトレースします。

ルールをローカルで定義することの欠点は、固定ターゲットが X-Ray サービスによって管理されるのではなく、レコーダーの各インスタンスによって個別に適用されることです。より多くのホストをデプロイすると、固定レートが乗算され、記録されるデータ量の制御が難しくなります。

では AWS Lambda、サンプリングレートを変更することはできません。関数がインストルメント化されたサービスによって呼び出された場合、そのサービスによってサンプリングされたリクエストを生成した呼び出しは Lambda によって記録されます。アクティブなトレースが有効で、トレースヘッダーが存在しない場合、Lambda はサンプリングを決定します。

バックアップルールを規定するには、NewCentralizedStrategyWithFilePath を使用して JSON ファイルのローカルサンプリングを指定します。

Example main.go – ローカルサンプリングルール

```
s, _ := sampling.NewCentralizedStrategyWithFilePath("sampling.json") // path to local
 sampling json
xray.Configure(xray.Config{SamplingStrategy: s})
```

ローカルルールのみを使用するには、NewLocalizedStrategyFromFilePath を使用して JSON ファイルのローカルサンプリングを指定します。

Example main.go – サンプリングを無効にする

```
s, _ := sampling.NewLocalizedStrategyFromFilePath("sampling.json") // path to local
 sampling json
xray.Configure(xray.Config{SamplingStrategy: s})
```

ログ記録

> **Note**
>
> xray.Config{} フィールド LogLevel と LogFormat は、バージョン 1.0.0-rc.10 以降では非推奨です。

X-Ray は、ログ記録に次のインターフェイスを使用します。デフォルトのロガーは、LogLevelInfo 以上で stdout に書き込みます。

```
type Logger interface {
 Log(level LogLevel, msg fmt.Stringer)
}

const (
 LogLevelDebug LogLevel = iota + 1
 LogLevelInfo
 LogLevelWarn
 LogLevelError
)
```

Example **io.Writer** に書き込み

```
xray.SetLogger(xraylog.NewDefaultLogger(os.Stderr, xraylog.LogLevelError))
```

環境変数

X-Ray SDK for Goの設定 SDK は次の変数をサポートしています。

- AWS_XRAY_CONTEXT_MISSING – 計測されたコードが、セグメントが開いていないときにデータを記録しようとした場合に例外をスローするには、RUNTIME_ERROR に設定します。

有効な値

- RUNTIME_ERROR— ランタイム例外をスローします。
- LOG_ERROR – エラーをログ記録して続行します (デフォルト)。
- IGNORE_ERROR – エラーを無視して続行します。

オープン状態のリクエストがない場合、または新しいスレッドを発生させるコードで、スタートアップコードに実装されたクライアントを使用しようとした場合に発生する可能性がある、セグメントまたはサブセグメントの欠落に関連するエラー。

- AWS_XRAY_TRACING_NAME – SDK がセグメントに使用するサービス名を設定します。
- AWS_XRAY_DAEMON_ADDRESS – X-Ray デーモン リスナーのホストとポートを設定します。デフォルトでは、SDK は、トレースデータをに送信します127.0.0.1:2000。この変数は、デーモンを次のように構成している場合に使用します。別のポートでリッスンするまたは、別のホストで実行されている場合。

環境変数は、コードで設定される同等の値を上書きします。

設定の使用

X-Ray SDK for Go を Configure を使用して設定することもできます。Configureは、1 つの引数、Config オブジェクトと次のオプションフィールドを使用します。

DaemonAddr

この文字列は X-Ray デーモン リスナーのホストとポートを指定します。指定しない場合、X-Ray は AWS_XRAY_DAEMON_ADDRESS 環境変数の値を使用します。この値が設定されていない場合は、「127.0.0.1:2000」を使用します。

ServiceVersion

この文字列は、サービスのバージョンを指定します。指定されていない場合、X-Ray は空の文字列 (「」) を使用します。

SamplingStrategy

この SamplingStrategy オブジェクトは、どのアプリケーションコールをトレースするかを指定します。指定しない場合、X-Ray は LocalizedSamplingStrategy で定義された戦略を取る xray/resources/DefaultSamplingRules.json を使用します。

StreamingStrategy

このStreamingStrategyオブジェクトは、 が true をRequiresStreaming返すときにセグメントをストリーミングするかどうかを指定します。指定しない場合、X-Ray は、サブセグメントの数が 20 を超える場合、サンプリングされたセグメントをストリーミングする DefaultStreamingStrategy を使用します。

ExceptionFormattingStrategy

この ExceptionFormattingStrategy オブジェクトは、さまざまな例外を処理する方法を指定します。指定しない場合、X-Ray は、タイプ DefaultExceptionFormattingStrategy の XrayError、エラーメッセージ、およびスタックトレースを持つ error を使用します。

X-Ray SDK for Goを使用して受信する HTTP リクエストの計測を行う

X-Ray SDK を使用して、アプリケーションが、Amazon EC2 AWS Elastic Beanstalk、または Amazon ECS の EC2 インスタンスで処理する受信 HTTP リクエストをトレースできます。

xray.Handler を使用して受信 HTTP リクエストを計測します。X-Ray SDK for Go は、標準の Go ライブラリ http.Handler インターフェイスを xray.Handler クラスに実装して、ウェブリクエストをインターセプトします。xray.Handler クラスは、リクエストのコンテキストを使って、提供された http.Handler を xray.Capture でラップし、必要に応じてレスポンスヘッダーを設定し、HTTP 固有のトレースフィールドを設定します。

このクラスを使用して HTTP リクエストとレスポンスを処理すると、X-Ray SDK for Go はサンプリングされた要求ごとにセグメントを作成します。このセグメントには、時間、メソッド、HTTP リクエストの処理などが含まれます。追加の計測により、このセグメントでサブセグメントが作成されます。

> **Note**
>
> AWS Lambda関数では、Lambda は、サンプリングされた各リクエストのセグメントを作成します。詳細については、「AWS Lambda および AWS X-Ray」を参照してください。

次の例は、ポート8000 でリクエストをインターセプトし、レスポンスとして「Hello!」 を返します。任意のアプリケーションでセグメント myApp と計測呼び出しを作成します。

Example main.go

```
func main() {
  http.Handle("/", xray.Handler(xray.NewFixedSegmentNamer("MyApp"),
 http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
    w.Write([]byte("Hello!"))
  })))

  http.ListenAndServe(":8000", nil)
}
```

各セグメントには、サービスマップ内のアプリケーションを識別する名前があります。セグメントの名前は静的に指定することも、受信リクエストのホストヘッダーに基づいて動的に名前を付けるように SDK を設定することもできます。動的ネーミングでは、リクエスト内のドメイン名に基づいてトレースをグループ化でき、名前が予想されるパターンと一致しない場合 (たとえば、ホストヘッダーが偽造されている場合) 、デフォルト名を適用できます。

> **転送されたリクエスト**
>
> ロードバランサーまたは他の仲介者がアプリケーションにリクエストを転送する場合、X-Ray は、クライアントの IP をIP パケットの送信元 IP からではなく、リクエストのX-Forwarded-Forヘッダーから取得します。転送されたリクエストについて記録されたクライアント IP は偽造される可能性があるため、信頼されるべきではありません。

リクエストが転送されると、それを示す追加フィールドが SDK によってセグメントに設定されます。セグメントのフィールド x_forwarded_for が true に設定されている場合、クライアント IP が X-Forwarded-For HTTP リクエストのヘッダーから取得されます。

ハンドラーは、次の情報が含まれる http ブロックを使用して、各受信リクエスト用にセグメントを作成します。

- HTTP メソッド – GET、POST、PUT、DELETE、その他。
- クライアントアドレス – リクエストを送信するクライアントの IP アドレス。
- レスポンスコード – 完了したリクエストの HTTP レスポンスコード。
- タイミング – 開始時間 (リクエストが受信された時間) および終了時間 (レスポンスが送信された時間)。
- ユーザーエージェント — リクエストからのuser-agent

- コンテンツの長さ — レスポンスからの `content-length`。

セグメント命名ルールの設定

AWS X-Rayはサービス名を使用してアプリケーションを識別し、他のアプリケーション、データベース、外部 API、およびAWSアプリケーションが使用するリソースと区別します。X-Ray SDK が受信リクエストのセグメントを生成すると、アプリケーションのサービス名がセグメントの名前フィールドに記録されます。

X-Ray SDK では、HTTP リクエストヘッダーのホスト名の後にセグメントの名前を指定できます。ただし、このヘッダーは偽造され、サービスマップに予期しないノードが発生する可能性があります。偽造されたホストヘッダーを持つリクエストによって SDK がセグメントの名前を間違えないようにするには、受信リクエストのデフォルト名を指定する必要があります。

アプリケーションが複数のドメインのリクエストを処理する場合、動的ネーミングストラテジーを使用してセグメント名にこれを反映するように SDK を設定できます。動的ネーミングストラテジーにより、SDK は予想されるパターンに一致するリクエストにホスト名を使用し、そうでないリクエストにデフォルト名を適用できます。

たとえば、3 つのサブドメイン (`www.example.com`,`api.example.com`,および`static.example.com`) に対してリクエストを処理する単一のアプリケーションがあるとします。動的ネーミングストラテジーをパターン `*.example.com` で使用して、異なる名前を持つ各サブドメインのセグメントを識別することができます。結果的にはサービスマップ上に 3 つのサービスノードを作成することになります。アプリケーションがパターンと一致しないホスト名のリクエストを受信すると、指定したフォールバック名を持つ 4 番目のノードがサービスマップに表示されます。

すべてのリクエストセグメントに対して同じ名前を使用するには、前のセクションで示すとおり、ハンドラーを作成するときに、アプリケーションの名前を指定します。

> **Note**
>
> コードで定義したデフォルトのサービス名は、`AWS_XRAY_TRACING_NAME` 環境変数で上書きできます。

動的命名ルールは、ホスト名と一致するようパターンを定義し、HTTP リクエストのホスト名がパターンと一致しない場合はデフォルトの名前を使用します。セグメントに動的に名前を付けるに

は、NewDynamicSegmentNamer を使用して、一致させるデフォルトの名前とパターンを設定します。

Example main.go

リクエストのホスト名がパターン *.example.com と一致する場合は、そのホスト名を使用します。それ以外の場合は、MyApp を使用します。

```
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)
}
```

X-Ray AWS SDK for Go を使用した SDK 呼び出しのトレース

アプリケーションが AWS のサービス を呼び出してデータの保存、キューへの書き込み、または通知の送信を行う場合、X-Ray SDK for Go はサブセグメント の呼び出しダウンストリームを追跡します。これらのサービス (Amazon S3 バケットや Amazon SQS キューなど) 内でアクセスするトレースされた AWS のサービス および リソースは、X-Ray コンソールのトレースマップにダウンストリームノードとして表示されます。

AWS SDK クライアントをトレースするには、次の例に示すように、クライアントオブジェクトを xray.AWS() 呼び出しでラップします。

Example main.go

```
var dynamo *dynamodb.DynamoDB
func main() {
  dynamo = dynamodb.New(session.Must(session.NewSession()))
  xray.AWS(dynamo.Client)
}
```

次に、AWS SDK クライアントを使用する場合は、呼び出しメソッドの withContext バージョンを使用し、それをcontextハンドラーhttp.Requestに渡された オブジェクトから に渡します。

Example main.go – AWS SDK 呼び出し

```
func listTablesWithContext(ctx context.Context) {
```

```
  output := dynamo.ListTablesWithContext(ctx, &dynamodb.ListTablesInput{})
  doSomething(output)
}
```

すべてのサービスにおいて、X-Ray コンソールでコールされた API の名前を確認できます。サービスのサブセットの場合、X-Ray SDK はセグメントに情報を追加して、サービスマップでより細かく指定します。

たとえば、実装された DynamoDB クライアントでコールすると、SDK はテーブルをターゲットとするコールのセグメントにテーブル名を追加します。コンソールで、各テーブルはサービスマップ内に個別のノードとして表示され、テーブルをターゲットにしないコール用の汎用の DynamoDB ノードが表示されます。

Example 項目を保存するための DynamoDB に対するコールのサブセグメント

```
{
  "id": "24756640c0d0978a",
  "start_time": 1.480305974194E9,
  "end_time": 1.4803059742E9,
  "name": "DynamoDB",
  "namespace": "aws",
  "http": {
    "response": {
      "content_length": 60,
      "status": 200
    }
  },
  "aws": {
    "table_name": "scorekeep-user",
    "operation": "UpdateItem",
    "request_id": "UBQNSO5AEM8T4FDA4RQDEB94OVTDRVV4K4HIRGVJF66Q9ASUAAJG",
  }
}
```

名前付きリソースにアクセスしたとき、次のサービスをコールすると、サービスマップに追加のノードが作成されます。特定のリソースをターゲットとしないコールでは、サービスの汎用ノードが作成されます。

- Amazon DynamoDB – テーブル名
- Amazon Simple Storage Service – バケットとキー名
- Amazon Simple Queue Service – キュー名

X-Ray SDK for Go を使用してダウンストリーム HTTP ウェブサービスの呼び出しをトレースする

アプリケーションがマイクロサービスまたはパブリック HTTP API を呼び出すときは、次の例に示すように、`xray.Client` を使用してこれらの呼び出しを Go アプリケーションのサブセグメントとして計測できます。次の例に示すように、http-client は HTTP クライアントです。

クライアントは、提供された HTTP クライアントのシャローコピーを作成します。これは `http.DefaultClient` のデフォルトの `xray.RoundTripper` でラウンドトリップされます。

Example

<caption>main.go – HTTP クライアント</caption>

```
myClient := xray.Client(http-client)
```

<caption>main.go — ctxhttp ライブラリを使用したダウンストリームの HTTP 呼び出しのトレース</caption>

次の例では、ctxhttp ライブラリを使用して発信 HTTP コールをインストルメントします。`xray.Client`。`ctx`アップストリームコールから渡すことができます。これにより、既存のセグメントコンテキストが使用されることが保証されます。たとえば、X-Ray では Lambda 関数内に新しいセグメントを作成できないので、既存の Lambda セグメントコンテキストを使用する必要があります。

```
resp, err := ctxhttp.Get(ctx, xray.Client(nil), url)
```

X-Ray SDK for Goを使用して SQL クエリをトレースします

PostgreSQL または MySQL への SQL 呼び出しをトレースするには、次の例に示すように、`sql.Open` への `xray.SQLContext` 呼び出しに置き換えます。可能であれば、構成文字列の代わりに URL を使用します。

Example main.go

```
func main() {
  db, err := xray.SQLContext("postgres", "postgres://user:password@host:port/db")
  row, err := db.QueryRowContext(ctx, "SELECT 1") // Use as normal
}
```

X-Ray SDK for Goを使用してカスタムサブセグメントを生成する

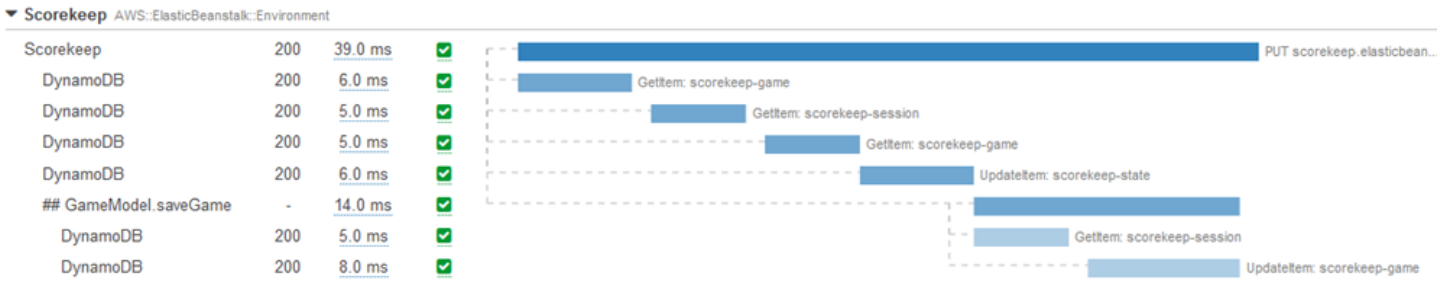
サブセグメントはトレースを拡張するセグメントリクエストを処理するために行われた作業の詳細を含む。計測済みクライアント内で呼び出しを行うたびに、X-Ray SDK によってサブセグメントに生成された情報が記録されます。追加のサブセグメントを作成して、他のサブセグメントをグループ化したり、コードセクションのパフォーマンスを測定したり、注釈とメタデータを記録したりできます。

`Capture` メソッドを使用して、関数の周囲にサブセグメントを作成します。

Example main.go – カスタムサブセグメント

```
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)
  })
```

次のスクリーンショットは、アプリケーション `saveGame` のトレースに `Scorekeep` サブセグメントがどのように表示されるかの例を示しています。



X-Ray SDK for Go を使用してセグメントに注釈とメタデータを追加する

アノテーションとメタデータを使用して、リクエスト、環境、またはアプリケーションに関する追加情報を記録できます。X-Ray SDK が作成するセグメントまたは作成するカスタムサブセグメントに、注釈およびメタデータを追加できます。

注釈は文字列、数値、またはブール値を使用したキーと値のペアです。注釈は、フィルタ式用にインデックス付けされます。注釈を使用して、コンソールでトレースをグループ化するため、またはGetTraceSummaries API を呼び出すときに使用するデータを記録します。

メタデータは、オブジェクトとリストを含む、任意のタイプの値を持つことができるキーバリューのペアですが、フィルタ式に使用するためにインデックスは作成されません。メタデータを使用してトレースに保存する追加のデータを記録しますが、検索で使用する必要はありません。

注釈とメタデータに加えて、セグメントにユーザー ID 文字列を記録することもできます。ユーザー ID はセグメントの個別のフィールドに記録され、検索用にインデックスが作成されます。

セクション

- X-Ray SDK for Goを使用して注釈を記録する
- X-Ray SDK for Goを使用してメタデータを記録する
- X-Ray SDK for Goを使用してユーザー ID の記録

X-Ray SDK for Goを使用して注釈を記録する

注釈を使用して、検索用にインデックスを作成するセグメントに情報を記録します。

注釈の要件

- キー — X-Ray アノテーションのキーには、最大 500 文字の英数字を使用できます。アンダースコア記号 (_) 以外のスペースや記号は使用できません。
- 値 — X-Ray アノテーションの値には、最大 1,000 文字の Unicode 文字を使用できます。
- 注釈の数 — 1 つのトレース 1 つにつき最大 50 個の注釈を使用できます。

注釈を記録するには、セグメントに関連付けるメタデータを含む文字列で AddAnnotation を呼び出します。

```
xray.AddAnnotation(key string, value interface{})
```

SDK は、セグメントドキュメントの annotations オブジェクトにキーと値のペアとして、注釈を記録します。同じキーで AddAnnotation を 2 回呼び出すと、同じセグメントに以前記録された値が上書きされます。

特定の値を持つ注釈のあるトレースを見つけるには、annotations.*key*フィルタ式の キーワードを使用します。

X-Ray SDK for Goを使用してメタデータを記録する

メタデータを使用して、検索用にインデックスを作成する必要のないセグメントに情報を記録します。

メタデータを記録するには、セグメントに関連付けるメタデータを含む文字列で AddMetadata を呼び出します。

```
xray.AddMetadata(key string, value interface{})
```

X-Ray SDK for Goを使用してユーザー ID の記録

リクエストセグメントにユーザー ID を記録して、リクエストを送信したユーザーを識別します。

ユーザー ID を記録するには

1. AWSXRay から現在のセグメントへの参照を取得します。

   ```
   import (
     "context"
     "github.com/aws/aws-xray-sdk-go/xray"
   )

   mySegment := xray.GetSegment(context)
   ```

2. リクエストを送信したユーザーの文字列 ID を使用して setUser を呼び出します。

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

ユーザー ID のトレースを見つけるには、userフィルタ式で、 キーワードを使用します。

を使用してアプリケーションを計測する Java

X-Ray にトレースを送信するようにJavaアプリケーションを計測するには、次の 2 つの方法があります。

- AWS Distro for OpenTelemetry Java – AWS Distro for OpenTelemetry Collector を介して相関メトリクスとトレースを Amazon CloudWatch、 AWS X-Ray、Amazon OpenSearch Service などの複数の AWS モニタリングソリューションに送信するための一連のオープンソースライブラリを提供するディス AWS トリビューション。

- AWS X-Ray SDK for Java – X-Ray デーモン を介してトレースを生成して X-Ray に送信するためのライブラリのセット。

詳細については、「AWS Distro for OpenTelemetry と X-Ray SDKs の選択」を参照してください。

AWS Distro for OpenTelemetry Java

AWS Distro for OpenTelemetry (ADOT) Java を使用すると、アプリケーションを一度計測し、相関のあるメトリクスとトレースを Amazon CloudWatch、AWS X-Ray、Amazon OpenSearch Service などの複数の AWS モニタリングソリューションに送信することができます。X-Ray を ADOT で使用するには、X-Ray で使用できる OpenTelemetry SDK と、X-Ray で使用できる AWS Distro for OpenTelemetry Collector の 2 つのコンポーネントが必要です。ADOT Java には自動計測のサポートが含まれており、アプリケーションでコードを変更せずにトレースを送信できます。

開始するには、AWS Distro for OpenTelemetry Java ドキュメントを参照してください。

AWS Distro for OpenTelemetry と AWS X-Ray やその他の AWS のサービス の併用については、「AWS Distro for OpenTelemetry」または「AWS Distro for OpenTelemetry Documentation」を参照してください。

言語サポートと使用方法の詳細については、「AWS Observability on Github」を参照してください。

AWS X-Ray の SDK Java

X-Ray SDK for Java は、トレースデータを生成して X-Ray デーモンに送信するためのクラスとメソッドを提供するJavaウェブアプリケーション用のライブラリのセットです。トレースデータには、アプリケーションによって処理される受信 HTTP リクエストに関する情報、および AWS SDK、HTTP クライアント、または SQL データベースコネクタを使用してアプリケーションがダウンストリームサービスに対して行う呼び出しが含まれます。セグメントを手動で作成し、注釈およびメタデータにデバッグ情報を追加することもできます。

X-Ray SDK for Java は、オープンソースプロジェクトです。プロジェクトに従い、 で問題やプルリクエストを送信できます GitHub。github.com/aws/aws-xray-sdk-java

受信リクエストのトレースは、`AWSXRayServletFilter` をサーブレットフィルタとして追加することから開始します。サーブレットフィルタにより、セグメントが作成されます。セグメントが開いている間、SDK クライアントのメソッドを使用してセグメントに情報を追加し、サブセグメントを作成してダウンストリーム呼び出しをトレースできます。また、SDK では、セグメントが開いている間にアプリケーションがスローする例外を自動的に記録します。

リリース 1.3 以降では、Spring のアスペクト指向プログラミング (AOP) を使用してアプリケーションを計測できます。つまり、アプリケーションのランタイムにコードを追加 AWSすることなく、 で実行中にアプリケーションを計測できます。

次に、X-Ray SDK for Java を使用して、SDK Instrumentor サブモジュールをビルド設定に含めることで AWS SDK for Java クライアントを計測します。計測されたクライアントを使用してダウンストリーム AWS のサービス またはリソースを呼び出すたびに、SDK は呼び出しに関する情報をサブセグメントに記録します。サービス内でアクセスするリソースは、トレースマップにダウンストリームノードとして表示 AWS のサービス され、個々の接続でエラーやスロットリングの問題を特定するのに役立ちます。

へのダウンストリーム呼び出しをすべて計測したくない場合は AWS のサービス、Instrumentor サブモジュールを省略して、計測するクライアントを選択できます。 AWS SDK サービスクライアントに を追加してTracingHandler、個々のクライアントを計測します。

その他の X-Ray SDK for Java サブモジュールでは、HTTP ウェブ API および SQL データベースに対するダウンストリーム呼び出しを計測できます。Apache HTTP サブモジュールで X-Ray SDK for Java のバージョン HTTPClient と HTTPClientBuilder を使用する と、Apache HTTP クライアントを計測することができます。SQL クエリの計測には、データソースに SDK のインターセプターを追加します。

SDK を使用し始めたら、レコーダーやサーブレットフィルターを設定して、SDK の動作をカスタマイズしてみましょう。プラグインを追加して、アプリケーションを実行しているコンピューティングリソースに関するデータを記録したり、サンプリングルールを定義することでサンプリングの動作のカスタマイズしたり、アプリケーションログに SDK からの情報をより多くあるいは少なく表示するようにログレベルを設定できます。

アプリケーションが注釈やメタデータで行うリクエストや作業に関する追加情報を記録します。注釈は、フィルタ式で使用するためにインデックス化されたシンプルなキーと値のペアで、特定のデータが含まれているトレースを検索できます。メタデータのエントリは制約が緩やかで、JSON にシリアル化できるオブジェクトと配列全体を記録できます。

> **注釈とメタデータ**
>
> 注釈およびメタデータとは、X-Ray SDK を使用してセグメントに追加する任意のテキストです。注釈は、フィルタ式用にインデックス付けされます。メタデータはインデックス化されませんが、X-Ray コンソールまたは API を使用して raw セグメントで表示できます。X-Ray への読み取りアクセスを許可した人は誰でも、このデータを表示できます。

コードに多数の計測されたクライアントがある場合、単一のリクエストセグメントには計測されたクライアントで行われた呼び出しごとに 1 個の多数のサブセグメントを含めることができます。カスタムサブセグメントで、クライアント呼び出しをラップすることで、サブセグメントを整理してグループできます。関数全体またはコードの任意のセクションのサブセグメントを作成し、親セグメントにすべてのレコードを記述する代わりにサブセグメントにメタデータと注釈を記録できます。

サブモジュール

X-Ray SDK for Java は、Maven からダウンロードできます。X-Ray SDK for Java は、ユースケースごとにサブモジュールに分割され、部品表のバージョン管理に使用されます。

- `aws-xray-recorder-sdk-core` (必須) セグメントを作成して送信するための基本的な機能です。受信リクエストを計測する `AWSXRayServletFilter` が含まれています。
- `aws-xray-recorder-sdk-aws-sdk` – トレース AWS SDK for Java クライアントをリクエストハンドラーとして追加することで、クライアントで AWS のサービス 行われた への呼び出しを計測します。
- `aws-xray-recorder-sdk-aws-sdk-v2` – トレースクライアントをリクエストインターセプターとして追加することで、 AWS SDK for Java 2.2 以降のクライアントで AWS のサービス 行われた の呼び出しを計測します。
- `aws-xray-recorder-sdk-aws-sdk-instrumentor` – では`aws-xray-recorder-sdk-aws-sdk`、 はすべての AWS SDK for Java クライアントを自動的に計測します。
- `aws-xray-recorder-sdk-aws-sdk-v2-instrumentor` – では`aws-xray-recorder-sdk-aws-sdk-v2`、 はすべての AWS SDK for Java 2.2 以降のクライアントを自動的に計測します。
- `aws-xray-recorder-sdk-apache-http` - Apache HTTP クライアントを使用して行われるアウトバウンド HTTP 呼び出しを計測します。
- `aws-xray-recorder-sdk-spring` – Spring AOP Framework アプリケーション用のインターセプターを提供します。
- `aws-xray-recorder-sdk-sql-postgres` - JDBC を使用して PostgreSQL データベースに対して行われるアウトバウンド呼び出しを計測します。
- `aws-xray-recorder-sdk-sql-mysql` - JDBC を使用して MySQL データベースに対して行われるアウトバウンド呼び出しを計測します。
- `aws-xray-recorder-sdk-bom` - すべてのサブモジュールで使用するバージョンを指定するための部品表を提供します。
- `aws-xray-recorder-sdk-metrics` – 収集した X-Ray セグメントからサンプリングされていない Amazon CloudWatch メトリクスを公開します。

Maven または Gradle を使用してアプリケーションを構築する場合は、X-Ray SDK for Java をビルド設定に追加します。

SDK のクラスとメソッドのリファレンスドキュメントについては、AWS X-Ray 「 SDK for Java API Reference」を参照してください。

要件

X-Ray SDK for Java には、8 Java 以降の Servlet API 3、 AWS SDK、Jackson が必要です。

SDK は、コンパイル時および実行時に次のライブラリに依存します。

- AWS SDK for Javaバージョン 1.11.398 以降
- Servlet API 3.1.0

これらの依存関係は SDK の `pom.xml` ファイルで宣言され、Maven や Gradle を使用して構築すると自動的に含まれます。

X-Ray SDK for Java に含まれているライブラリを使用する場合、同梱されているバージョンを使用する必要があります。たとえば、すでに実行時に Jackson に依存し、その依存関係のためにデプロイメントに JAR ファイルを含めている場合、SDK JAR には Jackson ライブラリの独自のバージョンが含まれているため、その JAR ファイルを削除する必要があります。

依存関係管理

X-Ray SDK for Javaは、Maven から入手できます。

- グループ – `com.amazonaws`
- Artifact – `aws-xray-recorder-sdk-bom`
- バージョン – `2.11.0`

Maven を使用してアプリケーションを構築する場合は、SDK を依存関係として `pom.xml` ファイルに追加します。

Example pom.xml - 依存関係

```
<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>
```

Gradle の場合は、SDK をコンパイル時の依存関係として `build.gradle` ファイルに追加します。

Example build.gradle - 依存関係

```
dependencies {
  compile("org.springframework.boot:spring-boot-starter-web")
  testCompile("org.springframework.boot:spring-boot-starter-test")
  compile("com.amazonaws:aws-java-sdk-dynamodb")
  compile("com.amazonaws:aws-xray-recorder-sdk-core")
```

```
  compile("com.amazonaws:aws-xray-recorder-sdk-aws-sdk")
  compile("com.amazonaws:aws-xray-recorder-sdk-aws-sdk-instrumentor")
  compile("com.amazonaws:aws-xray-recorder-sdk-apache-http")
  compile("com.amazonaws:aws-xray-recorder-sdk-sql-postgres")
  compile("com.amazonaws:aws-xray-recorder-sdk-sql-mysql")
  testCompile("junit:junit:4.11")
}
dependencyManagement {
    imports {
        mavenBom('com.amazonaws:aws-java-sdk-bom:1.11.39')
        mavenBom('com.amazonaws:aws-xray-recorder-sdk-bom:2.11.0')
    }
}
```

Elastic Beanstalk を使用してアプリケーションをデプロイする場合、すべての依存関係を含んだ大きなアーカイブを構築してアップロードする代わりに、Maven または Gradle を使用してデプロイするたびにオンインスタンスで構築できます。Gradle の使用例については、サンプルアプリケーションを参照してください。

AWS X-Ray Java 用自動計測エージェント

Java 用 AWS X-Ray 自動計測エージェントは、最小限の開発労力で Java ウェブアプリケーションを計測するトレースソリューションです。エージェントは、サーブレットベースのアプリケーションと、サポートされているフレームワークおよびライブラリで作成されたエージェントのすべてのダウンストリームリクエストのトレースを可能にします。これには、ダウンストリーム Apache HTTP リクエスト、 AWS SDK リクエスト、および JDBC ドライバーを使用して作成された SQL クエリが含まれます。エージェントは、すべてのアクティブなセグメントとサブセグメントを含む X-Ray コンテキストをスレッド間で伝播します。X-Ray SDK のすべての設定と汎用性は、Java エージェントで引き続き使用できます。エージェントが最小限の労力で動作するように、適切なデフォルトが選択されました。

X-Ray エージェントソリューションは、サーブレットベースの要求応答 Java ウェブアプリケーションサーバーに最適です。アプリケーションが非同期フレームワークを使用している場合、または要求/応答サービスとして適切にモデル化されていない場合は、代わりに SDK を使用した手動計測を検討することをお勧めします。

X-Ray エージェントは、分散システム理解ツールキット (DiSCo) を使用して構築されます。DiSCo は、分散システムで使用できる Java エージェントを構築するためのオープンソースフレームワークです。X-Ray エージェントの使用について DiSCo を理解する必要はありませんが、プロジェクトの詳細については、「」のホームページ GitHubを参照してください。X-Ray エージェントも完全に

オープンソース化されています。ソースコードを表示したり、貢献したり、エージェントに関する問題を提起したりするには、 のリポジトリ GitHubにアクセスしてください。

サンプルアプリケーション

eb-java-scorekeep サンプルアプリケーションは、X-Ray エージェントで計測されるよう調整されています。このブランチにはサーブレットフィルターやレコーダー構成は含まれません。これらの機能はエージェントによって行われるためです。ローカルまたは AWS リソースを使用してアプリケーションを実行するには、サンプルアプリケーションの Readme ファイルに記載されている手順に従います。サンプルアプリを使用して X-Ray トレースを生成する方法は、サンプルアプリのチュートリアルに記載されています。

使用開始方法

自分のアプリケーションで X-Ray 自動計測 Java エージェントを使用するには、次の手順を実行します。

1. ご使用の環境で X-Ray デーモンを実行します。詳細については、X-Ray デーモンを参照してください。
2. エージェントの最新ディストリビューションをダウンロードします。アーカイブを解凍し、ファイルシステム内の場所を記録します。その内容は次のようになります。

```
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. 下記を含めるようアプリケーションの JVM 引数を変更します。それにより、エージェントが有効になります。該当する場合は、`-javaagent` 引数が `-jar` 引数の 前 に配置されていることを確認します。JVM 引数を変更するプロセスは、Java サーバーの起動に使用するツールやフレームワークによって異なります。具体的なガイダンスについては、サーバーフレームワークのドキュメントを参照してください。

```
-javaagent:/<path-to-disco>/disco-java-agent.jar=pluginPath=/<path-to-disco>/disco-
plugins
```

4. X-Ray コンソールでのアプリケーション名の表示方法を指定するには、`AWS_XRAY_TRACING_NAME` 環境変数または

com.amazonaws.xray.strategy.tracingName システムプロパティを設定します。名前が指定されていない場合は、デフォルト名が使用されます。

5. サーバーまたはコンテナを再起動します。着信要求とそのダウンストリーム呼び出しがトレースされるようになりました。期待した結果が表示されない場合は、「the section called “トラブルシューティング”」を参照してください。

構成

X-Ray エージェントは、ユーザー提供の外部の JSON ファイルによって設定されます。デフォルトでは、このファイルはユーザーのクラスパスのルート (たとえば、ユーザーの resources ディレクトリ) に xray-agent.jsonという名前で存在します。com.amazonaws.xray.configFile システムプロパティに、設定ファイルの絶対ファイルシステムパスを設定することで、設定ファイルのカスタムロケーションを設定できます。

次に、設定ファイルの例を示します。

```
{
    "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
}
```

設定仕様

次の表は、各プロパティの有効な値を説明しています。プロパティ名は大文字と小文字が区別されますが、キーは区別されません。環境変数とシステムプロパティで上書きできるプロパティの場合、優先順位の順序は常に環境変数、システムプロパティ、構成ファイルになります。上書きできるプロパティについては、環境変数 を参照してください。すべてのフィールドはオプションです。

プロパティ名	タイプ	有効値	説明	環境変数	システムプロパティ	デフォルト値
serviceName	文字列	任意の文字列	X-Ray コンソールに表示される計測済みサービス名。	AWS_XRAY_TRACING_NAME	com.amazonaws.xray.strategy.tracingName	XRayInstrumentedService
contextMissingStrategy	文字列	LOG_ERROR、IGNORE_ERROR	エージェントが X-Ray セグメントコンテキストを使用しようとするが、存在しないときに実行するアクション。	AWS_XRAY_CONTEXT_MISSING	com.amazonaws.xray.strategy。contextMissingStrategy	LOG_ERROR
DaemonAddress	文字列	フォーマットされた IP アドレスとポート、または TCP および UDP アドレスのリスト	X-Ray デーモンとの通信にエージェントが使用するアドレス。	AWS_XRAY_DAEMON_ADDRESS	com.amazonaws.xray.emitter.daemonAddress	127.0.0.1:2000
tracingEnabled	ブール値	True、False	X-Ray エージェントによる計測を有効にします。	AWS_XRAY_TRACING_ENABLED	com.amazonaws.xray.tracingEnabled	TRUE

プロパティ名	タイプ	有効値	説明	環境変数	システムプロパティ	デフォルト値
samplingStrategy	文字列	CENTRAL、LOCAL、NONE、ALL	エージェントが使用するサンプリング戦略。ALL はすべてのリクエストをキャプチャし、NONE はリクエストをキャプチャしません。サンプリングルールを参照してください。。	該当なし	該当なし	CENTRAL
traceIdInjectionプレフィックス	文字列	任意の文字列	ログに注入されたトレース ID の前に指定されたプレフィクスを含めます。	該当なし	該当なし	なし (空の文字列)

プロパティ名	タイプ	有効値	説明	環境変数	システムプロパティ	デフォルト値
samplingRulesManifest	文字列	絶対ファイルパス	ローカルサンプリング戦略のサンプリングルール、または中央戦略のフォールバックルールのソースとして使用されるカスタムサンプリングルールファイルへのパス。	該当なし	該当なし	DefaultSamplingRules.json
awsServiceHandlerマニフェスト	文字列	絶対ファイルパス	AWS SDK クライアントから追加情報を取得するためのカスタムパラメータ許可リストへのパス。	該当なし	該当なし	DefaultOperationParameterWhitelist.json

プロパティ名	タイプ	有効値	説明	環境変数	システムプロパティ	デフォルト値
awsSdkVersion	整数	1、2	使用している AWS SDK for Java のバージョン。`awsServiceHandlerManifest` も設定されていない場合は無視されます。	該当なし	該当なし	2
maxStackTrace長さ	整数	非負整数	トレースに記録するスタックトレースの最大行数。	該当なし	該当なし	50
streamingThreshold	整数	非負整数	少なくともこの数のサブセグメントが閉じられると、チャンクが大きすぎるのを避けるため out-of-band にデーモンにストリーミングされます。	該当なし	該当なし	100

プロパティ名	タイプ	有効値	説明	環境変数	システムプロパティ	デフォルト値
traceIdInjection	ブール値	True、False	ログ作成設定 に記述された依存関係や設定も追加されている場合、ログへの X-Ray トレース ID の注入を有効にします。それ以外の場合は、何もしません。	該当なし	該当なし	TRUE
pluginsEnabled	ブール値	True、False	運用している AWS 環境に関するメタデータを記録するプラグインを有効にします。プラグインを参照してください。	該当なし	該当なし	TRUE

プロパティ名	タイプ	有効値	説明	環境変数	システムプロパティ	デフォルト値
collectSqlQueries	ブール値	True、False	SQL クエリ文字列を SQL サブセグメントにベストエフォートベースで記録します。	該当なし	該当なし	FALSE
contextPropagation	ブール値	True、False	True の場合、スレッド間で X-Ray コンテキストを自動的に伝播します。それ以外の場合、は ThreadLocal を使用してコンテキストを保存し、スレッド間での手動伝播が必要です。	該当なし	該当なし	TRUE

ログ作成設定

X-Ray エージェントのログレベルは、X-Ray SDK for Java と同じ方法で設定できます。X-Ray SDK for Java でのログ作成設定については、ログ記録 を参照してください。

手動実装

エージェントの自動計測に加えて手動計測を実行する場合は、プロジェクトへの依存関係として X-Ray SDK を追加します。受信リクエストのトレース に記述した SDK のカスタムサーブレットフィルターは、X-Ray エージェントと互換性がないことに注意してください。

> **Note**
>
> エージェントを使用しながら、手動計測を実行するには、X-Ray SDK の最新バージョンを使用する必要があります。

Maven プロジェクトで作業している場合は、以下の依存関係を `pom.xml` ファイルに追加します。

```
<dependencies>
  <dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-xray-recorder-sdk-core</artifactId>
    <version>2.11.0</version>
  </dependency>
  </dependencies>
```

Gradle プロジェクトで作業している場合は、以下の依存関係を `build.gradle` ファイルに追加します。

```
implementation 'com.amazonaws:aws-xray-recorder-sdk-core:2.11.0'
```

通常の SDK と同様に、エージェントを使用しながら、注釈、メタデータ、ユーザー ID に加えて、カスタムサブセグメント を追加することができます。エージェントはスレッド間でコンテキストを自動的に伝播するため、マルチスレッドアプリケーションを操作するときにコンテキストを伝播するための回避策は必要ありません。

トラブルシューティング

エージェントは全自動計測を行うため、問題が発生した場合、根本原因を特定することが困難な場合があります。X-Ray エージェントが期待通りに動作しない場合は、以下の問題点と解決策を確認してください。X-Ray エージェントと SDK は Jakarta Commons Logging (JCL) を使用しています。ログ作成出力を表示するには、次の例 (`log4j-jcl` または `jcl-over-slf4j`) のように、JCL をログ作成バックエンドに接続するブリッジがクラスパスにあることを確認します。

問題: アプリケーションで Java エージェントを有効にしたが、X-Ray コンソールに何も表示されない

X-Ray デーモンは同じマシンで動作していますか？

そうでない場合は、X-Ray デーモンドキュメントを参照して設定します。

アプリケーションログに「X-Ray エージェントレコーダーの初期化」というメッセージが表示されますか？

アプリケーションにエージェントを正しく追加した場合、このメッセージはアプリケーションの起動時に、リクエストを受け取り始める前に INFO レベルでログに記録されます。このメッセージが表示されない場合、Java エージェントは Java プロセスで実行されていません。入力ミスがない状態で、すべてのセットアップ手順を正しく実行していることを確認してください。

アプリケーションログに、 AWS X-Ray 「コンテキスト欠落例外の抑制」のようなエラーメッセージがいくつか表示されていますか？

これらのエラーは、エージェントが AWS SDK リクエストや SQL クエリなどのダウンストリームリクエストを計測しようとしているが、エージェントが自動的にセグメントを作成できなかったために発生します。これらのエラーが多く見られる場合、エージェントはユースケースに最適なツールではない可能性がありますので、代わりに X-Ray SDK を使用した手動計測を検討することをお勧めします。また、X-Ray SDK の デバッグログ を有効にすると、コンテキスト欠落例外が発生している場所のスタックトレースを確認できます。コードのこれらの部分をカスタムセグメントでラップできます。これにより、これらのエラーを解決できます。ダウンストリームリクエストをカスタムセグメントでラップする例については、スタートアップコードの計測のサンプルコードを参照してください。

問題: 期待していたセグメントの一部が、X-Ray コンソールに表示されない

アプリケーションでマルチスレッドを使用していますか？

作成される予定のセグメントがコンソールに表示されない場合は、アプリケーションのバックグラウンドスレッドが原因である可能性があります。アプリケーションが AWS SDK で Lambda 関数を 1 回だけ呼び出したり、HTTP エンドポイントを定期的にポーリングしたりするなど、「発動して忘れる」バックグラウンドスレッドを使用してタスクを実行する場合、スレッド間でコンテキストを伝達している間にエージェントが混乱する可能性があります。この問題が発生しているかどうかを確認するには、X-Ray SDK のデバッグログを有効にして、次のようなメッセージがないかどうかを確認してください。「進行中のサブセグメントの親となる、<NAME> という名前のセグメントを出力していません」 この問題を回避するには、サーバーが戻る前にバックグラウンドスレッドに参加して、そのスレッドで行われたすべての作業が記録されるようにします。または、エージェントの

contextPropagation の設定を false にすると、バックグラウンドスレッドでのコンテキスト伝播を無効にすることができます。この場合、カスタムセグメントをもつスレッドを手動で計測するか、それらのスレッドが生成するコンテキスト欠落例外を無視する必要があります。

サンプリングルールを設定しましたか？

X-Ray コンソールに一見ランダムな、または予期しないセグメントが表示される場合、あるいはコンソールに表示されるはずのセグメントが表示されない場合は、サンプリングの問題が発生している可能性があります。X-Ray エージェントは、X-Ray コンソールのルールを使用して、作成したすべてのセグメントに集中サンプリングを適用します。デフォルトのルールは、1 秒あたり 1 セグメントが、それ以降はセグメントの 5% がサンプリングされます。つまり、エージェントで迅速に作成されたセグメントはサンプリングされない可能性があります。これを解決するには、目的のセグメントを適切にサンプリングするカスタムサンプリングルールを X-Ray コンソールで作成する必要があります。詳細については、「 でサンプリングルールを設定する」を参照してくださいX-Ray コンソールの詳細。

X-Ray SDK for Java の設定

X-Ray SDK for Java には、グローバルレコーダーを提供する AWSXRay というクラスが含まれます。これは、コードの計測に使用できる TracingHandler です。グローバルレコーダーを設定して、受信 HTTP 呼び出しのセグメントを作成する AWSXRayServletFilter をカスタマイズできます。

セクション

- サービスプラグイン
- サンプリングルール
- ログ記録
- セグメントリスナー
- 環境変数
- システムプロパティ

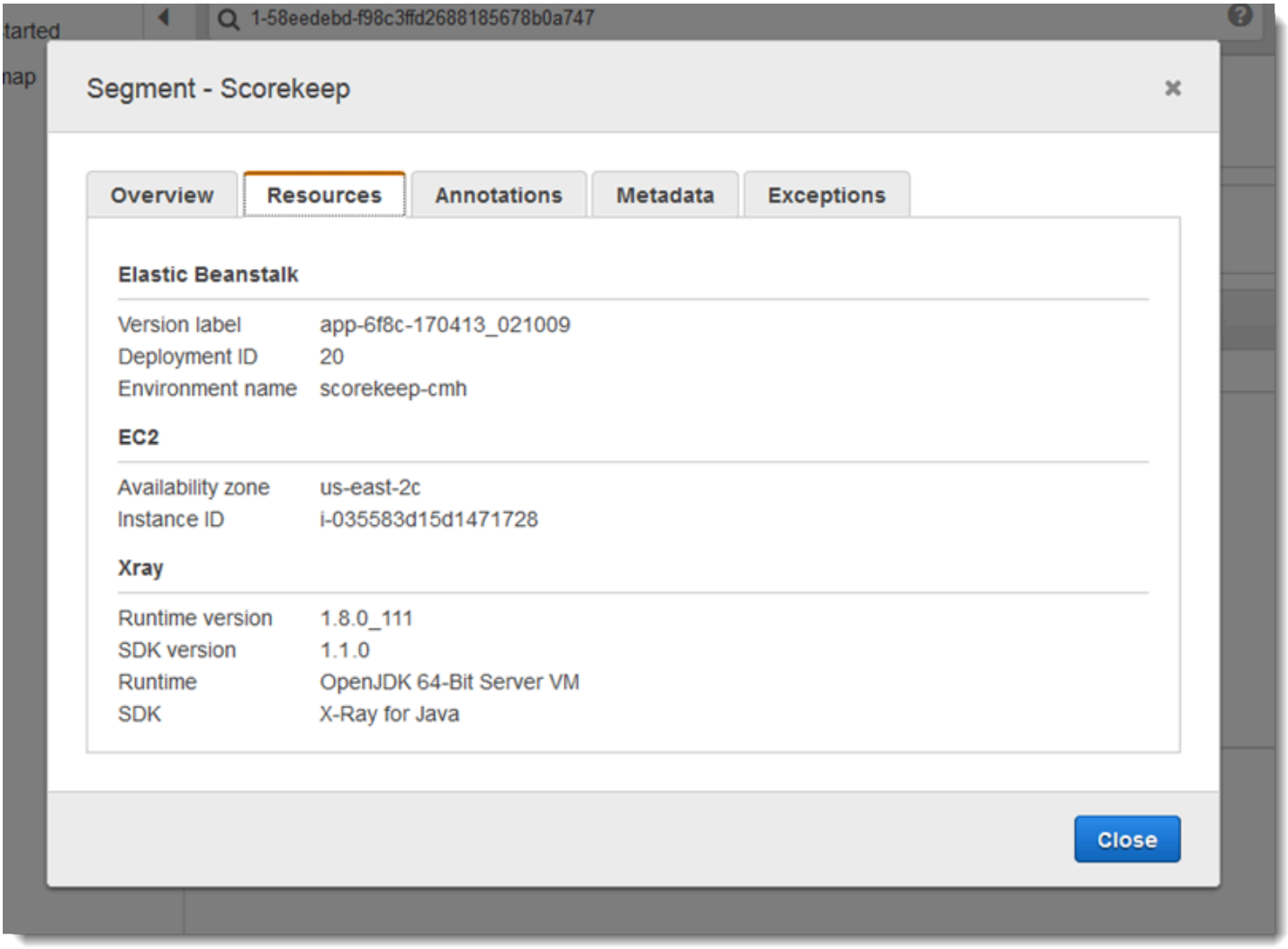
サービスプラグイン

pluginsを使用して、アプリケーションをホストしているサービスに関する情報を記録します。

プラグイン

- Amazon EC2 – インスタンス ID、アベイラビリティーゾーン、および CloudWatch ロググループEC2Pluginを追加します。

- ElasticBeanstalk– `ElasticBeanstalkPlugin`は、環境名、バージョンラベル、およびデプロイ ID を追加します。
- Amazon ECS —`ECSPlugin`は、コンテナ ID を追加します。
- Amazon EKS – コンテナ ID、クラスター名、ポッド ID、および CloudWatch ロググループ`EKSPlugin`を追加します。



プラグインを使用するには、`AWSXRayRecorderBuilder` で `withPlugin` を呼び出します。

Example src/main/java/scorekeep/WebConfig.java - レコーダー

```
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());
  }
}
```

SDK はプラグイン設定を使用して、セグメントのoriginフィールドを設定します。これは、アプリケーションを実行する AWS リソースのタイプを示します。複数のプラグインを使用する場合、SDK は次の解決順序を使用してオリジンを決定します。 ElasticBeanstalk > EKS > ECS > EC2。

サンプリングルール

SDK は X-Ray コンソールで定義したサンプリングルールを使用し、記録するリクエストを決定します。デフォルトルールでは、最初のリクエストを毎秒トレースし、X-Ray にトレースを送信するすべてのサービスで追加のリクエストの 5% をトレースします。X-Ray コンソールに追加のルールを作成するをクリックして、各アプリケーションで記録されるデータ量をカスタマイズします。

SDK は、定義された順序でカスタムルールを適用します。リクエストが複数のカスタムルールと一致する場合、SDK は最初のルールのみを適用します。

> **Note**
>
> SDK が X-Ray に到達してサンプリングルールを取得できない場合、SDK は毎秒最初に受信した最初のリクエストのデフォルトのローカルルールに戻り、ホストあたりの追加リクエストの 5% に戻ります。これは、ホストがサンプリング API を呼び出す権限を持っていない場合や、SDK によって行われる API 呼び出しの TCP プロキシとして機能する X-Ray デーモンに接続できない場合に発生します。

JSON ドキュメントからサンプリングルールをロードするように SDK を設定することもできます。SDK は、X-Ray サンプリングが利用できない場合のバックアップとしてローカルルールを使用することも、ローカルルールを排他的に使用することもできます。

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
  }
}
```

この例では、1 つのカスタムルールとデフォルトルールを定義します。カスタムルールでは、5 パーセントのサンプリングレートが適用され、/api/move/以下のパスに対してトレースするリクエストの最小数はありません。デフォルトのルールでは、1秒ごとの最初のリクエストおよび追加リクエストの 10 パーセントをトレースします。

ルールをローカルで定義することの欠点は、固定ターゲットが X-Ray サービスによって管理されるのではなく、レコーダーの各インスタンスによって個別に適用されることです。より多くのホストをデプロイすると、固定レートが乗算され、記録されるデータ量の制御が難しくなります。

では AWS Lambda、サンプリングレートを変更することはできません。関数がインストルメント化されたサービスによって呼び出された場合、そのサービスによってサンプリングされたリクエストを生成した呼び出しは Lambda によって記録されます。アクティブなトレースが有効で、トレースヘッダーが存在しない場合、Lambda はサンプリングを決定します。

Spring でバックアップルールを提供するには、設定クラスの `CentralizedSamplingStrategy` にグローバルレコーダーを設定します。

Example src/main/java/myapp/WebConfig.java - レコーダー設定

```
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());
}
```

Tomcat の場合、ServletContextListener を拡張するリスナーを追加し、デプロイ記述子にリスナーを登録します。

Example src/com/myapp/web/Startup.java

```
import com.amazonaws.xray.AWSXRay;
import com.amazonaws.xray.AWSXRayRecorderBuilder;
import com.amazonaws.xray.plugins.EC2Plugin;
import com.amazonaws.xray.strategy.sampling.LocalizedSamplingStrategy;

import java.net.URL;
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;

public class Startup implements ServletContextListener {

    @Override
    public void contextInitialized(ServletContextEvent event) {
        AWSXRayRecorderBuilder builder =
 AWSXRayRecorderBuilder.standard().withPlugin(new EC2Plugin());

        URL ruleFile = Startup.class.getResource("/sampling-rules.json");
```

```
        builder.withSamplingStrategy(new CentralizedSamplingStrategy(ruleFile));

        AWSXRay.setGlobalRecorder(builder.build());
    }

    @Override
    public void contextDestroyed(ServletContextEvent event) { }
}
```

Example WEB-INF/web.xml

```
...
  <listener>
    <listener-class>com.myapp.web.Startup</listener-class>
  </listener>
```

ローカルルールのみを使用するには、CentralizedSamplingStrategy を LocalizedSamplingStrategy に置き換えます。

```
builder.withSamplingStrategy(new LocalizedSamplingStrategy(ruleFile));
```

ログ記録

デフォルトでは、SDK はアプリケーションログに ERROR-レベルのメッセージを出力します。SDK でデバッグレベルのログを有効にすると、より詳細なログをアプリケーションログファイルに出力できます。有効なログレベルは、DEBUG、INFO、WARN、ERROR、FATAL です。FATAL ログレベルは、SDK が致命的なレベルでログを記録しないため、すべてのログメッセージを消去します。

Example application.properties

logging.level.com.amazonaws.xray プロパティを使用してログレベルを設定します。

```
logging.level.com.amazonaws.xray = DEBUG
```

デバッグログを使用して問題を識別します。たとえば、「サブセグメントを手動で生成する」場合にサブセグメントが閉じない問題などです。

ログへのトレース ID の挿入

ログステートメントに現在の完全修飾トレース ID を公開するには、マップされた診断コンテキスト (MDC) に ID を挿入できます。SegmentListener インターフェイスを使用して、セグメントライ

フサイクルイベント中に X-Ray レコーダーからメソッドが呼び出されます。セグメントまたはサブセグメントが開始されると、修飾トレース ID がキー AWS-XRAY-TRACE-ID と共に MDC に挿入されます。そのセグメントが終了すると、キーは MDC から削除されます。これにより、トレース ID が使用中のログ記録ライブラリに公開されます。サブセグメントが終了すると、その親 ID が MDC に挿入されます。

Example 完全修飾トレース ID

完全修飾 ID は TraceID@EntityID として表されます

```
1-5df42873-011e96598b447dfca814c156@541b3365be3dafc3
```

この機能は、 AWS X-Ray SDK for Java で計測された Java アプリケーションで動作し、次のログ記録設定をサポートします。

- Logback バックエンドを使用する SLF4J フロントエンド API
- Log4J2 バックエンドを使用する SLF4J フロントエンド API
- Log4J2 バックエンドを使用する Log4J2 フロントエンド API

各フロントエンドと各バックエンドのニーズについては、以下のタブを参照してください。

SLF4J Frontend

1. 以下の Maven 依存関係をプロジェクトに追加します。

   ```
   <dependency>
       <groupId>com.amazonaws</groupId>
       <artifactId>aws-xray-recorder-sdk-slf4j</artifactId>
       <version>2.11.0</version>
   </dependency>
   ```

2. AWSXRayRecorder を構築するときに withSegmentListener メソッドを含めます。これにより、SegmentListener クラスが追加されて、SLF4J MDC に新しいトレース ID が自動的に挿入されるようになります。

   SegmentListener は、ログステートメントのプレフィクスを設定するためのパラメータとしてオプションの文字列を取ります。プレフィクスは、次の方法で設定できます。

   - なし – デフォルトの AWS-XRAY-TRACE-ID プレフィックスを使用します。

- 空 – 空の文字列を使用します (例:"")。
- カスタム – 文字列で定義されているカスタムプレフィックスを使用します。

Example **AWSXRayRecorderBuilder** ステートメント

```
AWSXRayRecorderBuilder builder = AWSXRayRecorderBuilder
        .standard().withSegmentListener(new SLF4JSegmentListener("CUSTOM-
PREFIX"));
```

Log4J2 front end

1. 以下の Maven 依存関係をプロジェクトに追加します。

```
<dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-xray-recorder-sdk-log4j</artifactId>
    <version>2.11.0</version>
</dependency>
```

2. AWSXRayRecorder を構築するときに withSegmentListener メソッドを含めます。これにより、SegmentListener クラスが追加されて、SLF4J MDC に新しい完全修飾トレース ID が自動的に挿入されるようになります。

   SegmentListener は、ログステートメントのプレフィクスを設定するためのパラメータとしてオプションの文字列を取ります。プレフィクスは、次の方法で設定できます。

   - なし – デフォルトの AWS-XRAY-TRACE-ID プレフィックスを使用します。
   - 空 – 空の文字列 (例: "") を使用して、プレフィックスを削除します。
   - カスタム – 文字列で定義されているカスタムプレフィックスを使用します。

   Example **AWSXRayRecorderBuilder** ステートメント

```
AWSXRayRecorderBuilder builder = AWSXRayRecorderBuilder
        .standard().withSegmentListener(new Log4JSegmentListener("CUSTOM-
PREFIX"));
```

Logback backend

トレース ID をログイベントに挿入するには、各ログ記録ステートメントを書式設定するロガーの PatternLayout を変更する必要があります。

1. patternLayout が設定されている場所を見つけます。これは、プログラムで行うことも、XML 設定ファイルを使用して行うこともできます。詳細については、「Logback の設定」を参照してください。
2. 以降のログ記録ステートメントにトレース ID を挿入するには、patternLayout の任意の場所に %X{AWS-XRAY-TRACE-ID} を挿入します。%X{} は、MDC から提供されたキーを使用して値を取得することを示します。Logback PatternLayouts の の詳細については、「」を参照してくださいPatternLayout。

Log4J2 backend

1. patternLayout が設定されている場所を見つけます。これは、プログラムで行うことも、XML、JSON、YAML、またはプロパティ形式で記述された設定ファイルを使用して行うこともできます。

   設定ファイルを使用した Log4J2 の設定の詳細については、「設定」を参照してください。

   プログラムによる Log4J2 の設定の詳細については、「プログラムによる設定」を参照してください。
2. 以降のログ記録ステートメントにトレース ID を挿入するには、PatternLayout の任意の場所に %X{AWS-XRAY-TRACE-ID} を挿入します。%X{} は、MDC から提供されたキーを使用して値を取得することを示します。Log4J2 PatternLayouts の の詳細については、「パターンレイアウト」を参照してください。

トレース ID の挿入の例

以下に示しているのは、トレース ID を含むように変更された PatternLayout 文字列です。トレース ID は、スレッド名 (%t) の後、ログレベル (%-5p) の前に出力されます。

Example ID を挿入した **PatternLayout**

```
%d{HH:mm:ss.SSS} [%t] %X{AWS-XRAY-TRACE-ID} %-5p %m%n
```

AWS X-Ray は、簡単に解析できるように、ログステートメントにキーとトレース ID を自動的に出力します。以下に示しているのは、変更した PatternLayout を使用したログステートメントです。

Example ID を挿入したログステートメント

```
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
```

ログメッセージ自体はパターン %m に格納され、ロガーを呼び出すときに設定されます。

セグメントリスナー

セグメントリスナーは、AWSXRayRecorder によって生成されたセグメントの開始と終了などのライフサイクルイベントをインターセプトするためのインターフェイスです。セグメントリスナーイベント関数の実装では、onBeginSubsegment で作成された場合にすべてのサブセグメントに同じ注釈を追加したり、afterEndSegment を使用して各セグメントがデーモンに送信された後にメッセージをログに記録したりします。または、beforeEndSubsegment を使用して SQL インターセプタによって送信されたクエリを記録し、サブセグメントが SQL クエリを表しているかどうかを確認して、そうであればメタデータを追加します。

SegmentListener 関数の完全なリストについては、AWS X-Ray Recorder SDK for Java API のドキュメントを参照してください。

次の例は、onBeginSubsegment での作成時にすべてのサブセグメントに一貫性のある注釈を追加し、afterEndSegment を使用して各セグメントの最後にログメッセージを出力する方法を示しています。

Example MySegmentListener.java

```
import com.amazonaws.xray.entities.Segment;
import com.amazonaws.xray.entities.Subsegment;
import com.amazonaws.xray.listeners.SegmentListener;

public class MySegmentListener implements SegmentListener {
    .....

    @Override
    public void onBeginSubsegment(Subsegment subsegment) {
        subsegment.putAnnotation("annotationKey", "annotationValue");
```

```
    }

    @Override
    public void afterEndSegment(Segment segment) {
        // Be mindful not to mutate the segment
        logger.info("Segment with ID " + segment.getId());
    }
}
```

このカスタムセグメントリスナーは、AWSXRayRecorder を構築するときに参照されます。

Example AWSXRayRecorderBuilder ステートメント

```
AWSXRayRecorderBuilder builder = AWSXRayRecorderBuilder
        .standard().withSegmentListener(new MySegmentListener());
```

環境変数

環境変数を使用して、X-Ray SDK for Java を設定できます。SDK は次の変数をサポートしています。

- AWS_XRAY_CONTEXT_MISSING – 計測されたコードが、セグメントが開いていないときにデータを記録しようとした場合に例外をスローするには、RUNTIME_ERROR に設定します。

  有効な値

  - RUNTIME_ERROR— ランタイム例外をスローします。
  - LOG_ERROR – エラーをログ記録して続行します (デフォルト)。
  - IGNORE_ERROR – エラーを無視して続行します。

  オープン状態のリクエストがない場合、または新しいスレッドを発生させるコードで、スタートアップコードに実装されたクライアントを使用しようとした場合に発生する可能性がある、セグメントまたはサブセグメントの欠落に関連するエラー。
- AWS_XRAY_DAEMON_ADDRESS – X-Ray デーモンリスナーのホストとポートを設定します。デフォルトでは、SDK はトレースデータ (UDP) とサンプリング (TCP) の両方に127.0.0.1:2000を使用します。この変数は、デーモンを次のように構成している場合に使用します。別のポートでリッスンするまたは、別のホストで実行されている場合。

  [形式]

  - 同じポート – *address*:*port*

  - 異なるポート – `tcp:address:port udp:address:port`
- `AWS_LOG_GROUP` - ロググループの名前をアプリケーションに関連付けられたロググループに設定します。ロググループがアプリケーションと同じ AWS アカウントとリージョンを使用している場合、X-Ray はこの指定されたロググループを使用してアプリケーションのセグメントデータを自動的に検索します。ロググループの詳細については、「ロググループとストリームの使用」を参照してください。
- `AWS_XRAY_TRACING_NAME` – SDK がセグメントに使用するサービス名を設定します。サーブレットフィルタのセグメント命名ルールで設定したサービス名を上書きします。

環境変数は、同等の「システムプロパティ」と、コードで設定される値を上書きします。

システムプロパティ

システムプロパティは、「環境変数」に代わる JVM 固有の代替的な方法として使用できます。SDK では、以下のプロパティをサポートしています。

- `com.amazonaws.xray.strategy.tracingName` – `AWS_XRAY_TRACING_NAME` と同等です。
- `com.amazonaws.xray.emitters.daemonAddress` – `AWS_XRAY_DAEMON_ADDRESS` と同等です。
- `com.amazonaws.xray.strategy.contextMissingStrategy` – `AWS_XRAY_CONTEXT_MISSING` と同等です。

環境変数と同等の環境変数のいずれも設定されている場合は、環境変数の値が使用されます。どちらのメソッドでも、コードで設定される値は上書きされます。

X-Ray SDK for Java を使用して受信リクエストをトレースします。

X-Ray SDK を使用して、アプリケーションが Amazon EC2 の EC2 インスタンス AWS Elastic Beanstalk、または Amazon ECS で処理する受信 HTTP リクエストをトレースできます。 Amazon EC2

`Filter` を使用して受信 HTTP リクエストを計測します。X-Ray サーブレットフィルターをアプリケーションに追加すると、X-Ray SDK for Java によってサンプリングされた各リクエストのセグメントが作成されます。このセグメントには、時間、メソッド、HTTP リクエストの処理などが含まれます。追加の計測により、このセグメントでサブセグメントが作成されます。

> **Note**
>
> AWS Lambda 関数の場合、Lambda はサンプリングされたリクエストごとにセグメントを作成します。詳細については、「AWS Lambda および AWS X-Ray」を参照してください。

各セグメントには、サービスマップ内のアプリケーションを識別する名前があります。セグメントの名前は静的に指定することも、受信リクエストのホストヘッダーに基づいて動的に名前を付けるように SDK を設定することもできます。動的ネーミングでは、リクエスト内のドメイン名に基づいてトレースをグループ化でき、名前が予想されるパターンと一致しない場合 (たとえば、ホストヘッダーが偽造されている場合) 、デフォルト名を適用できます。

> **転送されたリクエスト**
>
> ロードバランサーまたは他の仲介者がアプリケーションにリクエストを転送する場合、X-Ray は、クライアントの IP をIP パケットの送信元 IP からではなく、リクエストのX-Forwarded-Forヘッダーから取得します。転送されたリクエストについて記録されたクライアント IP は偽造される可能性があるため、信頼されるべきではありません。

リクエストが転送されると、それを示す追加フィールドが SDK によってセグメントに設定されます。セグメントのフィールド `x_forwarded_for` が `true` に設定されている場合、クライアント IP が HTTP リクエストの `X-Forwarded-For` ヘッダーから取得されます。

メッセージハンドラーは、次の情報が含まれる `http` ブロックを使用して、各受信リクエスト用にセグメントを作成します。

- HTTP メソッド – GET、POST、PUT、DELETE、その他。
- クライアントアドレス – リクエストを送信するクライアントの IP アドレス。
- レスポンスコード – 完了したリクエストの HTTP レスポンスコード。
- タイミング – 開始時間 (リクエストが受信された時間) および終了時間 (レスポンスが送信された時間)。
- ユーザーエージェント — リクエストからの`user-agent`
- コンテンツの長さ — レスポンスからの`content-length`

セクション

- トレースフィルタをアプリケーション (Tomcat) に追加する
- トレースフィルタをアプリケーション (Spring) に追加する
- セグメント命名ルールの設定

トレースフィルタをアプリケーション (Tomcat) に追加する

Tomcat の場合は、プロジェクトの `<filter>` ファイルに `web.xml` を追加します。`fixedName` パラメーターを使用して、サービス名を指定し、着信リクエスト用に作成されたセグメントに適用します。

Example WEB-INF/web.xml - Tomcat

```
<filter>
  <filter-name>AWSXRayServletFilter</filter-name>
  <filter-class>com.amazonaws.xray.javax.servlet.AWSXRayServletFilter</filter-class>
  <init-param>
    <param-name>fixedName</param-name>
    <param-value>MyApp</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>AWSXRayServletFilter</filter-name>
  <url-pattern>*</url-pattern>
</filter-mapping>
```

トレースフィルタをアプリケーション (Spring) に追加する

Spring の場合は、`WebConfig` クラスに `Filter` を追加します。セグメント名を文字列として `AWSXRayServletFilter` コンストラクタに渡します。

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

```
package myapp;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Bean;
import javax.servlet.Filter;
import com.amazonaws.xray.javax.servlet.AWSXRayServletFilter;

@Configuration
public class WebConfig {
```

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

セグメント命名ルールの設定

AWS X-Ray は、サービス名を使用してアプリケーションを識別し、アプリケーションが使用する他のアプリケーション、データベース、外部 APIs、および AWS リソースと区別します。X-Ray SDK が受信リクエストのセグメントを生成すると、アプリケーションのサービス名がセグメントの名前フィールドに記録されます。

X-Ray SDK では、HTTP リクエストヘッダーのホスト名の後にセグメントの名前を指定できます。ただし、このヘッダーは偽造され、サービスマップに予期しないノードが発生する可能性があります。偽造されたホストヘッダーを持つリクエストによって SDK がセグメントの名前を間違えないようにするには、受信リクエストのデフォルト名を指定する必要があります。

アプリケーションが複数のドメインのリクエストを処理する場合、動的ネーミングストラテジーを使用してセグメント名にこれを反映するように SDK を設定できます。動的ネーミングストラテジーにより、SDK は予想されるパターンに一致するリクエストにホスト名を使用し、そうでないリクエストにデフォルト名を適用できます。

たとえば、3 つのサブドメイン (`www.example.com`,`api.example.com`,および`static.example.com`) に対してリクエストを処理する単一のアプリケーションがあるとします。動的ネーミングストラテジーをパターン `*.example.com` で使用して、異なる名前を持つ各サブドメインのセグメントを識別することができます。結果的にはサービスマップ上に 3 つのサービスノードを作成することになります。アプリケーションがパターンと一致しないホスト名のリクエストを受信すると、指定したフォールバック名を持つ 4 番目のノードがサービスマップに表示されます。

すべてのリクエストセグメントに対して同じ名前を使用するには、前のセクションで示すとおり、サーブレットフィルタを初期化するときに、アプリケーションの名前を指定します。これは、 を呼び出してコン`AWSXRayServletFilter`ストラクターに`SegmentNamingStrategy.fixed()`渡すSegmentNamingStrategyことで固定 を作成するのと同じ効果があります。

> **Note**
>
> コードで定義したデフォルトのサービス名は、AWS_XRAY_TRACING_NAME 環境変数で上書きできます。

動的な命名戦略は、ホスト名と一致するようパターンを定義し、HTTP リクエストのホスト名がパターンと一致しない場合はデフォルトの名前を使用します。Tomcat で動的にセグメントに命名するには、dynamicNamingRecognizedHosts および dynamicNamingFallbackName を使用して、パターンとデフォルト名をそれぞれ定義します。

Example WEB-INF/web.xml - 動的名前付けのサーブレットフィルタ

```
<filter>
  <filter-name>AWSXRayServletFilter</filter-name>
  <filter-class>com.amazonaws.xray.javax.servlet.AWSXRayServletFilter</filter-class>
  <init-param>
    <param-name>dynamicNamingRecognizedHosts</param-name>
    <param-value>*.example.com</param-value>
  </init-param>
  <init-param>
    <param-name>dynamicNamingFallbackName</param-name>
    <param-value>MyApp</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>AWSXRayServletFilter</filter-name>
  <url-pattern>*</url-pattern>
</filter-mapping>
```

Spring の場合は、 SegmentNamingStrategyを呼び出して動的 を作成しSegmentNamingStrategy.dynamic()、AWSXRayServletFilterコンストラクタに渡します。

Example src/main/java/myapp/WebConfig.java - 動的命名を使用したサーブレットフィルター

```
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"));
  }
}
```

X-Ray AWS SDK for Java を使用した SDK 呼び出しのトレース

アプリケーションが AWS のサービス を呼び出してデータの保存、キューへの書き込み、または通知の送信を行うと、X-Ray SDK for Java はサブセグメント のダウンストリームの呼び出しを追跡します。これらのサービス (Amazon S3 バケットや Amazon SQS Amazon SQS キューなど) 内でアクセスするトレースされた AWS のサービス およびリソースは、X-Ray コンソールのトレースマップにダウンストリームノードとして表示されます。

aws-sdk および aws-sdk-instrumentor サブモジュールをビルドに含めると、X-Ray SDK for Java では自動的にすべての AWS SDK クライアントを計測します。Instrumentor サブモジュールを含めない場合は、一部のクライアントを計測して他を除外できます。

個々のクライアントを計測するには、ビルドからaws-sdk-instrumentorサブモジュールを削除し、サービスのクライアントビルダーを使用して AWS SDK クライアントTracingHandlerに XRayClient として を追加します。

たとえば、AmazonDynamoDB を計測するには、トレースハンドラーを AmazonDynamoDBClientBuilder に渡します。

Example MyModel.java - DynamoDB クライアント

```
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();
...
```

すべてのサービスにおいて、X-Ray コンソールでコールされた API の名前を確認できます。サービスのサブセットの場合、X-Ray SDK はセグメントに情報を追加して、サービスマップでより細かく指定します。

たとえば、実装された DynamoDB クライアントでコールすると、SDK はテーブルをターゲットとするコールのセグメントにテーブル名を追加します。コンソールで、各テーブルはサービスマップ内に個別のノードとして表示され、テーブルをターゲットにしないコール用の汎用の DynamoDB ノードが表示されます。

Example 項目を保存するための DynamoDB に対するコールのサブセグメント

```
{
  "id": "24756640c0d0978a",
  "start_time": 1.480305974194E9,
  "end_time": 1.4803059742E9,
  "name": "DynamoDB",
  "namespace": "aws",
  "http": {
    "response": {
      "content_length": 60,
      "status": 200
    }
  },
  "aws": {
    "table_name": "scorekeep-user",
    "operation": "UpdateItem",
    "request_id": "UBQNSO5AEM8T4FDA4RQDEB94OVTDRVV4K4HIRGVJF66Q9ASUAAJG",
  }
}
```

名前付きリソースにアクセスしたとき、次のサービスをコールすると、サービスマップに追加のノードが作成されます。特定のリソースをターゲットとしないコールでは、サービスの汎用ノードが作成されます。

- Amazon DynamoDB – テーブル名
- Amazon Simple Storage Service – バケットとキー名
- Amazon Simple Queue Service – キュー名

AWS SDK for Java 2.2 以降 AWS のサービス で へのダウンストリーム呼び出しを計測するには、ビルド設定からaws-xray-recorder-sdk-aws-sdk-v2-instrumentorモジュールを省略できます。その代わりに、aws-xray-recorder-sdk-aws-sdk-v2 module を含め、TracingInterceptor で設定して個々のクライアントを実装します。

Example AWS SDK for Java 2.2 以降 - インターセプターのトレース

```
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();
//...
```

X-Ray SDK for Java を使用してダウンストリーム HTTP ウェブサービスの呼び出しをトレースする

アプリケーションがマイクロサービスまたはパブリック HTTP API に呼び出しを実行する場合に、X-Ray SDK for Java の HttpClient バージョンを使用してこれらの呼び出しを計測し、API をダウンストリームサービスとしてサービスグラフに追加できます。

X-Ray SDK for Java には、送信 HTTP 呼び出しを計測するために Apache HttpComponents に相当するクラスの代わりに使用できる DefaultHttpClientおよび HttpClientBuilder クラスが含まれています。

- 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

これらのライブラリは、aws-xray-recorder-sdk-apache-httpサブモジュールにあります。

既存のインポートステートメントを X-Ray の該当部分に置き換えてすべてのクライアントを計測するか、クライアントを初期化する際に完全修飾名を使用して特定のクライアントを計測できます。

Example HttpClientBuilder

```
import com.fasterxml.jackson.databind.ObjectMapper;
import org.apache.http.HttpEntity;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.util.EntityUtils;
import com.amazonaws.xray.proxies.apache.http.HttpClientBuilder;
...
  public String randomName() throws IOException {
    CloseableHttpClient httpclient = HttpClientBuilder.create().build();
    HttpGet httpGet = new HttpGet("http://names.example.com/api/");
    CloseableHttpResponse response = httpclient.execute(httpGet);
    try {
      HttpEntity entity = response.getEntity();
      InputStream inputStream = entity.getContent();
      ObjectMapper mapper = new ObjectMapper();
      Map<String, String> jsonMap = mapper.readValue(inputStream, Map.class);
      String name = jsonMap.get("name");
      EntityUtils.consume(entity);
      return name;
    } finally {
      response.close();
    }
  }
```

ダウンストリームウェブ API に対する呼び出しを計測すると、X-Ray SDK for Java は HTTP リクエストおよびレスポンスに関する情報を含むセグメントを記録します。X-Ray はサブセグメントを使用してリモート API の推測セグメントを生成します。

Example ダウンストリーム HTTP 呼び出しのサブセグメント

```
{
  "id": "004f72be19cddc2a",
  "start_time": 1484786387.131,
  "end_time": 1484786387.501,
  "name": "names.example.com",
  "namespace": "remote",
  "http": {
```

```
    "request": {
      "method": "GET",
      "url": "https://names.example.com/"
    },
    "response": {
      "content_length": -1,
      "status": 200
    }
  }
}
```

Example ダウンストリーム HTTP 呼び出しの推定セグメント

```
{
  "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
}
```

X-Ray SDK for Java での SQL クエリのトレース

SQL インターセプター

X-Ray SDK for Java JDBC インターセプターをデータソース設定に追加して、SQL データベースのクエリを計測します。

- PostgreSQL – `com.amazonaws.xray.sql.postgres.TracingInterceptor`
- MySQL – `com.amazonaws.xray.sql.mysql.TracingInterceptor`

これらのインターセプタは、それぞれaws-xray-recorder-sql-postgres と aws-xray-recorder-sql-mysql サブモジュールにあります。これらは、Tomcat の接続プールと互換性がある `org.apache.tomcat.jdbc.pool.JdbcInterceptor` を実装します。

> **Note**
>
> SQL インターセプタは、セキュリティ上の目的で SQL クエリ自体をサブセグメント内に記録しません。

Spring の場合は、プロパティファイルのインターセプターを追加し、スプリングブートの `DataSourceBuilder` を使用して、データソースを構築します。

Example **`src/main/java/resources/application.properties`** - PostgreSQL JDBC Interceptor

```
spring.datasource.continue-on-error=true
spring.jpa.show-sql=false
spring.jpa.hibernate.ddl-auto=create-drop
spring.datasource.jdbc-interceptors=com.amazonaws.xray.sql.postgres.TracingInterceptor
spring.jpa.database-platform=org.hibernate.dialect.PostgreSQL94Dialect
```

Example **`src/main/java/myapp/WebConfig.java`** - データソース

```
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();
  }
...
}
```

Tomcat の場合、X-Ray SDK for Java クラスを参照する JDBC データソースの `setJdbcInterceptors` を呼び出します。

Example **`src/main/myapp/model.java`** - データソース

```
import org.apache.tomcat.jdbc.pool.DataSource;
...
DataSource source = new DataSource();
source.setUrl(url);
source.setUsername(user);
source.setPassword(password);
source.setDriverClassName("com.mysql.jdbc.Driver");
source.setJdbcInterceptors("com.amazonaws.xray.sql.mysql.TracingInterceptor;");
```

Tomcat JDBC データソースライブラリは、X-Ray SDK for Java に含まれていますが、使用するドキュメントへの指定された依存関係として宣言できます。

Example **`pom.xml`** - JDBC Data Source

```
<dependency>
  <groupId>org.apache.tomcat</groupId>
  <artifactId>tomcat-jdbc</artifactId>
  <version>8.0.36</version>
  <scope>provided</scope>
</dependency>
```

ネイティブ SQL トレースデコレータ

- [aws-xray-recorder-sdk-sql] を依存関係に追加します。

- データベースのデータソース、接続、またはステートメントを修飾します。

```
dataSource = TracingDataSource.decorate(dataSource)
connection = TracingConnection.decorate(connection)
statement = TracingStatement.decorateStatement(statement)
preparedStatement = TracingStatement.decoratePreparedStatement(preparedStatement,
 sql)
callableStatement = TracingStatement.decorateCallableStatement(callableStatement,
 sql)
```

X-Ray SDK for Java を使用したカスタムサブセグメントの生成

サブセグメントは、トレースの セグメント をリクエストを処理するために行われた作業の詳細で拡張します。計測済みクライアント内で呼び出しを行うたびに、X-Ray SDK によってサブセグメントに生成された情報が記録されます。追加のサブセグメントを作成して、他のサブセグメントをグループ化したり、コードセクションのパフォーマンスを測定したり、注釈とメタデータを記録したりできます。

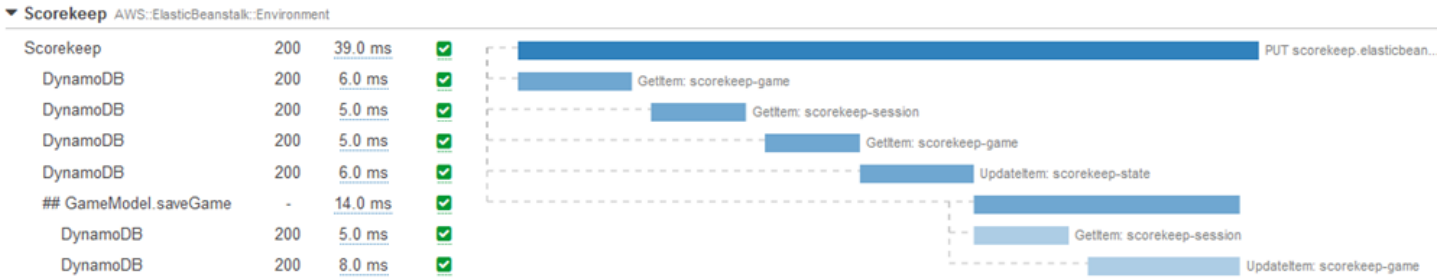
サブセグメントを管理するには、beginSubsegment および endSubsegment メソッドを使用します。

Example GameModel.java - カスタムサブセグメント

```
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();
    }
```

```
  }
```

この例では、サブセグメント内のコードは、セッションモデルのメソッドを使用して DynamoDB からゲームのセッションをロードし、 AWS SDK for Javaの DynamoDB マッパーを使用してゲームを保存します。このコードをサブセグメントにラップすることで、呼び出しが `Save Game` サブセグメントの DynamoDB の子としてコンソールのトレースビューに表示されます。



サブセグメントのコードがチェック例外をスローした場合は、`try` ブロックにコードをラップして、`finally` ブロックで `AWSXRay.endSubsegment()` を呼び出し、常にサブセグメントが閉じられるようにします。サブセグメントが閉じていない場合は、親セグメントが完了できず、X-Ray に送信されません。

チェック例外をスローしないコードの場合は、コードを Lambda 関数として `AWSXRay.CreateSubsegment` に渡すことができます。

Example Lambda 関数のサブセグメント

```
import com.amazonaws.xray.AWSXRay;

AWSXRay.createSubsegment("getMovies", (subsegment) -> {
    // function code
});
```

セグメントまたは別のサブセグメント内にサブセグメントを作成する場合、X-Ray SDK for Java によってその ID が生成され、開始時刻と終了時刻が記録されます。

Example サブセグメントとメタデータ

```
"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"
    }
  },
```

非同期やマルチスレッドのプログラミングでは、非同期実行中に X-Ray コンテキストが変更されることがあるため、サブセグメントを endSubsegment() メソッドに手動で渡して正しく閉じるようにする必要があります。親セグメントが閉じられた後に非同期サブセグメントが閉じられた場合、このメソッドはセグメント全体を X-Ray デーモンに自動的にストリームします。

Example 非同期サブセグメント

```
@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();
}
```

X-Ray SDK for Java を使用してセグメントに注釈とメタデータを追加する

注釈とメタデータを使用して、リクエスト、環境、またはアプリケーションに関する追加情報を記録できます。X-Ray SDK が作成するセグメントまたは作成するカスタムサブセグメントに、注釈およびメタデータを追加できます。

注釈は文字列、数値、またはブール値を使用したキーと値のペアです。注釈は、フィルタ式用にインデックス付けされます。注釈を使用して、コンソールでトレースをグループ化するため、または`GetTraceSummaries` API を呼び出すときに使用するデータを記録します。

メタデータは、オブジェクトとリストを含む、任意のタイプの値を持つことができるキーバリューのペアですが、フィルタ式に使用するためにインデックスは作成されません。メタデータを使用してトレースに保存する追加のデータを記録しますが、検索で使用する必要はありません。

注釈とメタデータに加えて、セグメントにユーザー ID 文字列を記録することもできます。ユーザー ID はセグメントの個別のフィールドに記録され、検索用にインデックスが作成されます。

セクション

- X-Ray SDK for Java での注釈の記録
- X-Ray SDK for Java を使用したメタデータの記録
- X-Ray SDK for Javaを使用したユーザー ID の記録

X-Ray SDK for Java での注釈の記録

注釈を使用して、検索用にインデックスを作成するセグメントまたはサブセグメントに情報を記録します。

注釈の要件

- キー – X-Ray 注釈のキーには、最大 500 文字の英数字を使用できます。アンダースコア記号 (_) 以外のスペースや記号は使用できません。
- 値 – X-Ray 注釈の値は、最大 1,000 文字の Unicode 文字を持つことができます。
- 注釈の数 — トレースごとに最大 50 個の注釈を使用できます。

注釈を記録するには

1. AWSXRay から現在のセグメントまたはサブセグメントの参照を取得します。

```
import com.amazonaws.xray.AWSXRay;
import com.amazonaws.xray.entities.Segment;
...
Segment document = AWSXRay.getCurrentSegment();
```

   または

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

2. 文字列キー、および、ブール値、数値、文字列値を使用して putAnnotation を呼び出します。

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

SDK は、セグメントドキュメントの annotations オブジェクトにキーと値のペアとして、注釈を記録します。同じキーで putAnnotation を 2 回呼び出すと、同じセグメントまたはサブセグメントに以前記録された値が上書きされます。

特定の値を持つ注釈のあるトレースを見つけるには、annotations.*key*フィルタ式の キーワードを使用します。

Example **src/main/java/scorekeep/GameModel.java** – 注釈とメタデータ

```
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();
    }
  }
```

X-Ray SDK for Java を使用したメタデータの記録

メタデータを使用して、検索用にインデックスを作成する必要のないセグメントまたはサブセグメントに情報を記録します。メタデータ値は、文字列、数値、ブール値、または JSON オブジェクトや JSON 配列にシリアル化できる任意のオブジェクトになります。

メタデータを記録するには

1. AWSXRay から現在のセグメントまたはサブセグメントの参照を取得します。

```
import com.amazonaws.xray.AWSXRay;
import com.amazonaws.xray.entities.Segment;
...
Segment document = AWSXRay.getCurrentSegment();
```

   または

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

2. 文字列名前空間、文字列キー、ブール値、数値、文字列値、オブジェクト値を使用して putMetadata を呼び出します。

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

   または

   キーと値だけを使用して putMetadata を呼び出します。

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

名前空間を指定しない場合、SDK は default を使用します。同じキーで putMetadata を 2 回呼び出すと、同じセグメントまたはサブセグメントに以前記録された値が上書きされます。

Example **src/main/java/scorekeep/GameModel.java** – 注釈とメタデータ

```
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();
    }
  }
```

X-Ray SDK for Javaを使用したユーザー ID の記録

リクエストセグメントにユーザー ID を記録して、リクエストを送信したユーザーを識別します。

ユーザー ID を記録するには

1. AWSXRay から現在のセグメントへの参照を取得します。

```
import com.amazonaws.xray.AWSXRay;
import com.amazonaws.xray.entities.Segment;
...
Segment document = AWSXRay.getCurrentSegment();
```

2. リクエストを送信したユーザーの文字列 ID を使用して setUser を呼び出します。

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

コントローラーで setUser を呼び出し、アプリケーションがリクエストの処理を開始するとすぐに、ユーザー ID を記録できます。ユーザー ID を設定するためだけにセグメントを使用する場合、呼び出しを 1 行で連鎖させることができます。

Example src/main/java/scorekeep/MoveController.java – ユーザー ID

```
import com.amazonaws.xray.AWSXRay;
...
  @RequestMapping(value="/{userId}", method=RequestMethod.POST)
  public Move newMove(@PathVariable String sessionId, @PathVariable String
 gameId, @PathVariable String userId, @RequestBody String move) throws
 SessionNotFoundException, GameNotFoundException, StateNotFoundException,
 RulesException {
    AWSXRay.getCurrentSegment().setUser(userId);
    return moveFactory.newMove(sessionId, gameId, userId, move);
  }
```

ユーザー ID のトレースを見つけるには、userフィルタ式で、 キーワードを使用します。

AWS X-Ray X-Ray SDK for Java の メトリクス

このトピックでは、 AWS X-Ray 名前空間、メトリクス、ディメンションについて説明します。X-Ray SDK for Java を使用して、収集した X-Ray セグメントからサンプリングされていない Amazon CloudWatch メトリクスを発行できます。これらのメトリクスは、セグメントの開始時間と終了時間、さらにエラー、障害、スロットリングのステータスフラグから取得されます。これらの追跡メトリクスを使用して、サブセグメント内の再試行と依存関係の問題を公開します。

CloudWatch はメトリクスリポジトリです。メトリクスは の基本的な概念 CloudWatch であり、時系列のデータポイントのセットを表します。ユーザー (または AWS のサービス) はメトリクスデータポイントを に発行 CloudWatch し、それらのデータポイントに関する統計を時系列データの順序付けられたセットとして取得します。

メトリクスは名前、名前空間、1 つ以上のディメンションで一意に定義されます。各データポイントには、タイムスタンプと、オプションとして測定単位があります。統計を要求した場合、返されるデータストリームは、名前空間、メトリクス名、ディメンションによって識別されます。

の詳細については CloudWatch、「Amazon ユーザーガイド CloudWatch 」を参照してください。

X-Ray CloudWatch メトリクス

ServiceMetrics/SDK 名前空間には、次のメトリクスが含まれます。

メトリクス	利用可能な統計情報	説明	単位
Latency	Average、Minimum、Maximum、Count	開始時間と終了時間の差。Average、Minimum、Maximum はすべて、オペレーション関連のレイテンシーを表します。Count は、呼び出し数を表します。	ミリ秒
ErrorRate	Average、Sum	4xx Client Error ステータスコードで失敗し、エラーになったリクエストの割合。	割合 (%)
FaultRate	Average、Sum	5xx Server Error ステータスコードで失敗し、障害になったトレースの割合。	割合 (%)
ThrottleRate	Average、Sum	429 ステータスコードを返すスロットリングトレースの割合。これは ErrorRate メトリクスのサブセットです。	割合 (%)
OkRate	Average、Sum	トレースされた後、OK ステータスコードになったリクエストの割合。	割合 (%)

X-Ray の CloudWatch ディメンション

次の表のディメンションを使用して、X-Ray 計測Javaアプリケーションに返されるメトリクスを絞り込みます。

ディメンション	説明
`ServiceType`	不明な場合、サービスのタイプ (`AWS::EC2::Instance` 、`NONE` など)。
`ServiceName`	サービスの正規名。

X-Ray CloudWatch メトリクスを有効にする

計測されたJavaアプリケーションでトレースメトリクスを有効にするには、次の手順を使用します。

トレースメトリクスを設定するには

1. `aws-xray-recorder-sdk-metrics` パッケージをApache Maven依存関係として追加します。詳細は、X-Ray SDK for Java サブモジュールを参照してください。
2. グローバルレコーダービルドの一部として新しい `MetricsSegmentListener()` を有効にします。

   Example src/com/myapp/web/Startup.java

```
import com.amazonaws.xray.AWSXRay;
import com.amazonaws.xray.AWSXRayRecorderBuilder;
import com.amazonaws.xray.plugins.EC2Plugin;
import com.amazonaws.xray.plugins.ElasticBeanstalkPlugin;
import com.amazonaws.xray.strategy.sampling.LocalizedSamplingStrategy;

@Configuration
public class WebConfig {
...
  static {
    AWSXRayRecorderBuilder builder = AWSXRayRecorderBuilder
                                        .standard()
                                        .withPlugin(new EC2Plugin())
                                        .withPlugin(new ElasticBeanstalkPlugin())
```

```
                                        .withSegmentListener(new
 MetricsSegmentListener());

    URL ruleFile = WebConfig.class.getResource("/sampling-rules.json");
    builder.withSamplingStrategy(new LocalizedSamplingStrategy(ruleFile));

    AWSXRay.setGlobalRecorder(builder.build());
  }
}
```

3. CloudWatch エージェントをデプロイして、Amazon Elastic Compute Cloud (Amazon EC2)、Amazon Elastic Container Service (Amazon ECS)、または Amazon Elastic Kubernetes Service (Amazon EKS) を使用してメトリクスを収集します。

   - Amazon EC2 を設定するには、CloudWatch 「 エージェントのインストール」を参照してください。
   - Amazon ECS を設定するには、「Container Insights を使用して Amazon ECS コンテナを監視する」を参照してください。
   - Amazon EKS を設定するには、「Amazon CloudWatch Observability EKS アドオン を使用して CloudWatch エージェントをインストールする」を参照してください。

4. CloudWatch エージェントと通信するように SDK を設定します。デフォルトでは、SDK はアドレス で CloudWatch エージェントと通信します`127.0.0.1`。環境変数または Java プロパティを `address:port` に設定することで、代替アドレスを設定できます。

   Example 環境変数

   ```
   AWS_XRAY_METRICS_DAEMON_ADDRESS=address:port
   ```

   Example Java のプロパティ

   ```
   com.amazonaws.xray.metrics.daemonAddress=address:port
   ```

設定を検証するには

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開きます。
2. [Metrics (メトリクス)] タブを開いて、メトリクスの到着を確認します。

3. （オプション) CloudWatch コンソールのログタブで、ServiceMetricsSDKロググループを開きます。ホストメトリクスに一致するログストリームを探し、ログメッセージを確認します。

マルチスレッドアプリケーションでのスレッド間のセグメントコンテキストの受け渡し

アプリケーションで新しいスレッドを作成すると、AWSXRayRecorder は現在のセグメントまたはサブセグメント Entity への参照を保持しません。計測されたクライアントを新しいスレッドで使用すると、SDK は存在しないセグメントへの書き込みを試み、 が発生しますSegmentNotFoundException。

開発中に例外がスローされないように、代わりにエラーをログに記録するように指示ContextMissingStrategyする を使用してレコーダーを設定できます。を使用してコードで戦略を設定することもSetContextMissingStrategy、環境変数またはシステムプロパティ を使用して同等のオプションを設定することもできます。

エラーに対処する 1 つの方法として、スレッドを開始するときに beginSegment を呼び出して新しいセグメントを使用する方法と、スレッドを終了するときに endSegment を使用する方法があります。これは、アプリケーションの起動時に実行されるコードのように、HTTP リクエストに応答して実行されないコードを実装する場合に機能します。

複数のスレッドを使用して着信リクエストを処理する場合は、現在のセグメントまたはサブセグメントを新しいスレッドに渡してグローバルレコーダーに渡すことができます。これにより、新しいスレッド内に記録された情報が、そのリクエストに関して記録された残りの情報と同じセグメントに関連付けられることが保証されます。新しいスレッドでセグメントが使用可能になると、そのセグメントのコンテキストにアクセスできる任意の実行可能なファイルを segment.run(() -> { ... }) メソッドで実行できるようになります。

例については、「実装されたクライアントをワーカースレッドで使用する」を参照してください。

非同期プログラミングでの X-Ray の使用

X-Ray SDK for Java は、 の非同期 Java プログラムで使用できますSegmentContextExecutors。は Executor インターフェイス SegmentContextExecutor を実装します。つまり、 のすべての非同期オペレーションに渡すことができますCompletableFuture。これにより、非同期オペレーションがそのコンテキスト内で正しいセグメントで実行されることが保証されます。

Example App.java の例: SegmentContextExecutor に渡す CompletableFuture

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

```
AWSXRay.beginSegment();

// ...

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

Spring による AOP と X-Ray SDK for Java

このトピックでは、X-Ray SDK および Spring Framework を使用して、コアロジックを変更せずにアプリケーションを計測する方法について説明します。つまり、 でリモートで実行されているアプリケーションを計測する非破壊的な方法が追加されました AWS。

Spring で AOP を有効にするには

1. Spring を設定します
2. トレースフィルターをアプリケーションに追加する
3. コードに注釈を付けるか、インターフェイスを実装します
4. アプリケーションで X-Ray を有効化します

Spring の設定

Maven または Gradle を使用して、AOP でアプリケーションを計測するように Spring を設定できます。

Maven を使用してアプリケーションを構築する場合は、次の依存関係を pom.xml ファイルに追加します。

```
<dependency>
     <groupId>com.amazonaws</groupId>
     <artifactId>aws-xray-recorder-sdk-spring</artifactId>
     <version>2.11.0</version>
</dependency>
```

Gradle の場合は、次の依存関係を build.gradle ファイルに追加します。

```
compile 'com.amazonaws:aws-xray-recorder-sdk-spring:2.11.0'
```

Spring Boot の設定

前のセクションで説明した Spring の依存関係に加えて、Spring Boot を使用している場合で、それがまだクラスパス上にない場合は、次の依存関係を追加します。

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'
```

トレースフィルターをアプリケーションに追加する

`WebConfig` のクラスに `Filter` を追加します。セグメント名を文字列として `AWSXRayServletFilter` コンストラクタに渡します。フィルターのトレースおよび受信リクエストの計測の詳細については、「X-Ray SDK for Java を使用して受信リクエストをトレースします。」を参照してください。

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

```
package myapp;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Bean;
import javax.servlet.Filter;
import com.amazonaws.xray.javax.servlet.AWSXRayServletFilter;

@Configuration
public class WebConfig {

  @Bean
  public Filter TracingFilter() {
    return new AWSXRayServletFilter("Scorekeep");
```

```
  }
}
```

Jakarta のサポート

Spring 6 の Enterprise Edition では、Javax ではなく Jakarta を使用しています。この新しい名前空間をサポートするために、X-Ray では独自の Jakarta 名前空間に存在するクラスの並列セットを作成しました。

フィルタークラスの場合は、`javax` を `jakarta` に置き換えてください。セグメント命名ルールを設定するときは、次の例のように、命名ルールクラス名の前に `jakarta` を追加してください。

```
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"));
    }
}
```

コードに注釈を付ける、またはインターフェイスを実装する

クラスには `@XRayEnabled` の注釈を付けるか、`XRayTraced` インターフェイスを実装する必要があります。これにより、X-Ray の計測のため、影響を受けた関数をラップするように AOP システムに伝えます。

アプリケーションでの X-Ray のアクティベーション

アプリケーションで X-Ray トレースを有効化するには、コードで次のメソッドをオーバーライドして、抽象クラス `BaseAbstractXRayInterceptor` を拡張する必要があります。

- `generateMetadata`—この関数では、現在の関数のトレースにアタッチされたメタデータをカスタマイズできます。デフォルトでは、実行中の関数のクラス名がメタデータに記録されます。追加情報が必要な場合は、さらにデータを追加できます。

- xrayEnabledClasses—この関数は空であり、空のままにしておく必要があります。これは、ラップするメソッドをインターセプターに指示するポイントカットのホストとして機能します。@XRayEnabled の注釈が付けられたどのクラスをトレースするか指定して、ポイントカットを定義します。次のステートメントは、@XRayEnabled という注釈が付けられたすべてのコントローラービーンをラップするようにインターセプターに伝えます。

```
@Pointcut(“@within(com.amazonaws.xray.spring.aop.XRayEnabled) && bean(*Controller)”)
```

プロジェクトで Spring Data JPA を使用している場合は、BaseAbstractXRayInterceptor ではなく AbstractXRayInterceptor から拡張することを検討してください。

例

次のコードは、抽象クラス BaseAbstractXRayInterceptorを示しています。

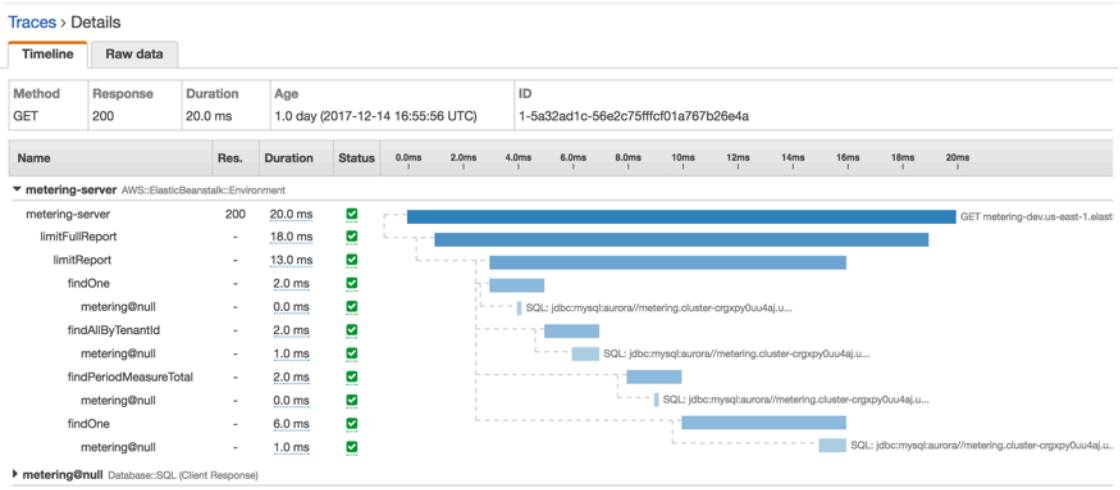
```
@Aspect
@Component
public class XRayInspector extends BaseAbstractXRayInterceptor {
    @Override
    protected Map<String, Map<String, Object>> generateMetadata(ProceedingJoinPoint
 proceedingJoinPoint, Subsegment subsegment) throws Exception {
        return super.generateMetadata(proceedingJoinPoint, subsegment);
    }

  @Override
  @Pointcut("@within(com.amazonaws.xray.spring.aop.XRayEnabled) && bean(*Controller)")

  public void xrayEnabledClasses() {}

}
```

次のコードは、X-Ray によって計測されるクラスです。

```
@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());
        }
    }
}
```

アプリケーションを正しく設定した場合は、コンソールの次のスクリーンショットに示すように、コントローラーからサービス呼び出しまで、アプリケーションの完全なコールスタックが表示されます。



を使用してアプリケーションを計測する Node.js

X-Ray にトレースを送信する Node.js アプリケーションを計測するには、次の 2 つの方法があります。

- AWS Distro for OpenTelemetry JavaScript – AWS Distro for OpenTelemetry Collector を介して相関メトリクスとトレースを Amazon 、 CloudWatch AWS X-Ray、Amazon OpenSearch Service などの複数の AWS モニタリングソリューションに送信するための一連のオープンソースライブラリを提供するディス AWS トリビューション。
- AWS X-Ray SDK for Node.js – X-Ray デーモン を介してトレースを生成して X-Ray に送信するためのライブラリのセット。

詳細については、「AWS Distro for OpenTelemetry と X-Ray SDKs の選択」を参照してください。

AWS Distro for OpenTelemetry JavaScript

AWS Distro for OpenTelemetry (ADOT) JavaScript を使用すると、アプリケーションを一度計測し、相関のあるメトリクスとトレースを Amazon CloudWatch、AWS X-Ray、Amazon OpenSearch Service などの複数の AWS モニタリングソリューションに送信することができます。X-Ray と AWS Distro for OpenTelemetry を使用するには、X-Ray で使用できる OpenTelemetry SDK と、X-Ray で使用できる AWS Distro for OpenTelemetry Collector の 2 つのコンポーネントが必要です。

開始するには、AWS Distro for OpenTelemetry JavaScript ドキュメントを参照してください。

> **Note**
>
> ADOT JavaScript は、すべてのサーバーサイドの Node.js アプリケーションでサポートされています。ADOT JavaScript はブラウザクライアントから X-Ray にデータをエクスポートできません。

AWS Distro for OpenTelemetry と AWS X-Ray やその他の AWS のサービス の併用については、「AWS Distro for OpenTelemetry」または「AWS Distro for OpenTelemetry Documentation」を参照してください。

言語サポートと使用方法の詳細については、「AWS Observability on Github」を参照してください。

AWS Node.js 用 X-Ray SDK

X-Ray SDK for Node.js は、Express ウェブアプリケーションと Node.js Lambda 関数用のライブラリです。トレースデータを作成して X-Ray デーモンに送信するためのクラスとメソッドを提供します。トレースデータには、アプリケーションによって処理される受信 HTTP リクエストに関する情報と、アプリケーションが AWS SDK または HTTP クライアントを使用してダウンストリームサービスに対して行う呼び出しが含まれます。

> **Note**
>
> X-Ray SDK for Node.js は、Node.js バージョン 14.x 以降でサポートされているオープンソースプロジェクトです。プロジェクトに従い、 で問題やプルリクエストを送信できます GitHub。github.com/aws/aws-xray-sdk-node

Express を使用する場合は、アプリケーションサーバーで SDK をミドルウェアとして追加し、受信リクエストをトレースします。ミドルウェアでは、トレース対象リクエストごとに「セグメント」を作成し、レスポンスが送信されるとセグメントを完了します。セグメントが開いている間、SDK クライアントのメソッドを使用してセグメントに情報を追加し、サブセグメントを作成してダウンストリーム呼び出しをトレースできます。また、SDK では、セグメントが開いている間にアプリケーションがスローする例外を自動的に記録します。

インストルメント済みアプリケーションまたはサービスによって呼び出される Lambda 関数の場合、Lambda は トレースヘッダー を読み取り、サンプリングされたリクエストを自動的にトレースします。その他の関数については、Lambda の設定 から受信リクエストのサンプリングとトレースを行うことができます。いずれの場合も、Lambda はセグメントを作成し、X-Ray SDK に提供します。

> **Note**
>
> Lambda では、X-Ray SDK はオプションです。関数でこれを使用しない場合、サービスマップには Lambda サービスのノードと Lambda 関数ごとに 1 つのノードが含まれます。SDK を追加することで、関数コードをインストルメントして、Lambda で記録された関数セグメントにサブセグメントを追加することができます。詳細については、「AWS Lambda および AWS X-Ray」を参照してください。

次に、X-Ray SDK for Node.js を使用して、Node.js クライアント JavaScript で の AWS SDK を計測します。計測されたクライアントを使用してダウンストリーム AWS のサービス またはリソースを呼び出すたびに、SDK は呼び出しに関する情報をサブセグメントに記録します。 AWS のサービス また、サービス内でアクセスするリソースは、トレースマップにダウンストリームノードとして表示され、個々の接続のエラーやスロットリングの問題を特定するのに役立ちます。

また、X-Ray SDK for Node.js では、HTTP ウェブ API と SQL クエリに対するダウンストリーム呼び出しの計測もできます。HTTP クライアントを SDK のキャプチャメソッドでラップして、送信 HTTP 呼び出しについての情報を記録します。SQL クライアントでは、データベースタイプのキャプチャメソッドを使用します。

ミドルウェアでは、受信リクエストにサンプリングルールを適用して、トレースするリクエストを決定します。X-Ray SDK for Node.js を設定して、サンプリング動作を調整したり、アプリケーションが実行される AWS コンピューティングリソースに関する情報を記録したりできます。

アプリケーションが注釈やメタデータで行うリクエストや作業に関する追加情報を記録します。注釈は、フィルタ式で使用するためにインデックス化されたシンプルなキーと値のペアで、特定のデータ

が含まれているトレースを検索できます。メタデータのエントリは制約が緩やかで、JSON にシリアル化できるオブジェクトと配列全体を記録できます。

> **注釈とメタデータ**
>
> 注釈およびメタデータとは、X-Ray SDK を使用してセグメントに追加する任意のテキストです。注釈は、フィルタ式用にインデックス付けされます。メタデータはインデックス化されませんが、X-Ray コンソールまたは API を使用して raw セグメントで表示できます。X-Ray への読み取りアクセスを許可した人は誰でも、このデータを表示できます。

コードに多数の計測されたクライアントがある場合、単一のリクエストセグメントには計測されたクライアントで行われた呼び出しごとに 1 個の多数のサブセグメントを含めることができます。カスタムサブセグメントで、クライアント呼び出しをラップすることで、サブセグメントを整理してグループできます。関数全体またはコードの任意のセクションのサブセグメントを作成し、親セグメントにすべてのレコードを記述する代わりにサブセグメントにメタデータと注釈を記録できます。

SDK のクラスとメソッドに関するリファレンスドキュメントについては、AWS X-Ray SDK for Node.js API リファレンスを参照してください。

要件

X-Ray SDK for Node.js には Node.js および次のライブラリが必要です。

- `atomic-batcher` – 1.0.2
- `cls-hooked` – 4.2.2
- `pkginfo` – 0.4.0
- `semver` – 5.3.0

NPM を使用して SDK をインストールするときに、SDK ではこれらのライブラリを引き出します。

AWS SDK クライアントをトレースするには、X-Ray SDK for Node.js には、Node.js JavaScript のの AWS SDK の最小バージョンが必要です。

- `aws-sdk` - 2.7.15

依存関係管理

The X-Ray SDK for Node.js は NPM から入手できます。

- パッケージ – aws-xray-sdk

ローカル開発の場合は、npm を使用してプロジェクトディレクトリに SDK をインストールします。

```
~/nodejs-xray$ npm install aws-xray-sdk
aws-xray-sdk@3.3.3
  ### aws-xray-sdk-core@3.3.3
  # ### @aws-sdk/service-error-classification@3.15.0
  # ### @aws-sdk/types@3.15.0
  # ### @types/cls-hooked@4.3.3
  # # ### @types/node@15.3.0
  # ### atomic-batcher@1.0.2
  # ### cls-hooked@4.2.2
  # # ### async-hook-jl@1.7.6
  # # # ### stack-chain@1.3.7
  # # ### emitter-listener@1.1.2
  # #   ### shimmer@1.2.1
  # ### semver@5.7.1
  ### aws-xray-sdk-express@3.3.3
  ### aws-xray-sdk-mysql@3.3.3
  ### aws-xray-sdk-postgres@3.3.3
```

--save オプションを使用して、SDK を依存関係としてアプリケーションの package.json に保存します。

```
~/nodejs-xray$ npm install aws-xray-sdk --save
aws-xray-sdk@3.3.3
```

アプリケーションに X-Ray SDK の依存関係と競合するバージョンの依存関係がある場合、互換性を確保するために両方のバージョンがインストールされます。詳細については、依存関係の解決に関する公式 NPM ドキュメンテーションを参照してください。

Node.js サンプル

AWS X-Ray SDK for Node.js を使用して、Node.js アプリケーションを通過するリクエスト end-to-end を表示します。

- の Node.js サンプルアプリケーション GitHub。

X-Ray SDK for Node.js の設定

X-Ray SDK for Node.js にプラグインを設定して、アプリケーションが実行されているサービスに関する情報が含めたり、デフォルトのサンプリング動作を変更したり、特定のパスに対するリクエストに適用されるサンプリングルールを追加したりできます。

セクション

- サービスプラグイン
- サンプリングルール
- ログ記録
- X-Ray デーモンのアドレス
- 環境変数

サービスプラグイン

`plugins`を使用して、アプリケーションをホストしているサービスに関する情報を記録します。

プラグイン

- Amazon EC2 – インスタンス ID、アベイラビリティーゾーン、および CloudWatch ロググループ`EC2Plugin`を追加します。
- ElasticBeanstalk– `ElasticBeanstalkPlugin`は、環境名、バージョンラベル、およびデプロイ ID を追加します。
- Amazon ECS —`ECSPlugin`は、コンテナ ID を追加します。

プラグインを使用するには、`config` メソッドを使用して X-Ray SDK for Node.js クライアントを設定します。

Example app.js - プラグイン

```
var AWSXRay = require('aws-xray-sdk');
AWSXRay.config([AWSXRay.plugins.EC2Plugin,AWSXRay.plugins.ElasticBeanstalkPlugin]);
```

SDK はプラグイン設定を使用して、セグメントのoriginフィールドを設定します。これは、アプリケーションを実行する AWS リソースのタイプを示します。複数のプラグインを使用する場合、SDK は次の解決順序を使用してオリジンを決定します。 ElasticBeanstalk > EKS > ECS > EC2。

サンプリングルール

SDK は X-Ray コンソールで定義したサンプリングルールを使用し、記録するリクエストを決定します。デフォルトルールでは、最初のリクエストを毎秒トレースし、X-Ray にトレースを送信するすべてのサービスで追加のリクエストの 5% をトレースします。X-Ray コンソールに追加のルールを作成するをクリックして、各アプリケーションで記録されるデータ量をカスタマイズします。

SDK は、定義された順序でカスタムルールを適用します。リクエストが複数のカスタムルールと一致する場合、SDK は最初のルールのみを適用します。

> **Note**
>
> SDK が X-Ray に到達してサンプリングルールを取得できない場合、SDK は毎秒最初に受信した最初のリクエストのデフォルトのローカルルールに戻り、ホストあたりの追加リクエストの 5% に戻ります。これは、ホストがサンプリング API を呼び出す権限を持っていない場合や、SDK によって行われる API 呼び出しの TCP プロキシとして機能する X-Ray デーモンに接続できない場合に発生します。

JSON ドキュメントからサンプリングルールをロードするように SDK を設定することもできます。SDK は、X-Ray サンプリングが利用できない場合のバックアップとしてローカルルールを使用することも、ローカルルールを排他的に使用することもできます。

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
  }
}
```

この例では、1 つのカスタムルールとデフォルトルールを定義します。カスタムルールでは、5 パーセントのサンプリングレートが適用され、/api/move/以下のパスに対してトレースするリクエストの最小数はありません。デフォルトのルールでは、1秒ごとの最初のリクエストおよび追加リクエストの 10 パーセントをトレースします。

ルールをローカルで定義することの欠点は、固定ターゲットが X-Ray サービスによって管理されるのではなく、レコーダーの各インスタンスによって個別に適用されることです。より多くのホストをデプロイすると、固定レートが乗算され、記録されるデータ量の制御が難しくなります。

では AWS Lambda、サンプリングレートを変更することはできません。関数がインストルメント化されたサービスによって呼び出された場合、そのサービスによってサンプリングされたリクエストを生成した呼び出しは Lambda によって記録されます。アクティブなトレースが有効で、トレースヘッダーが存在しない場合、Lambda はサンプリングを決定します。

バックアップルールを設定するには、setSamplingRulesの ファイルからサンプリングルールをロードするようX-Ray SDK for Node.js に指示します。

Example app.js - ファイルのサンプリングルール

```
var AWSXRay = require('aws-xray-sdk');
AWSXRay.middleware.setSamplingRules('sampling-rules.json');
```

コードのルールを定義し、オブジェクトとして setSamplingRules に渡すこともできます。

Example app.js - オブジェクトのサンプリングルール

```
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);
```

ローカルルールのみを使用するには、disableCentralizedSampling を呼び出します。

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

ログ記録

SDK からログ出力するには、AWSXRay.setLogger(logger) を呼び出します。logger は、標準のログ記録メソッド (warn、info、など) を提供するオブジェクトです。

デフォルトでは、SDK はコンソールオブジェクトの標準メソッドを使用してコンソールにエラーメッセージを記録します。組み込みロガーのログレベルは、AWS_XRAY_DEBUG_MODEまたはAWS_XRAY_LOG_LEVEL環境変数を使って設定できます。有効なログレベル値の一覧については、「環境変数」を参照してください。

ログに別の形式または宛先を指定したい場合は、次に示すように、SDK に独自のロガーインターフェイスの実装を提供できます。このインタフェースを実装するあらゆるオブジェクトを使用できます。つまり、Winston など、多くのロギングライブラリを使用して SDK に直接渡すことができるということです。

Example app.js - ログ記録

```
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]);
```

他の設定方法を実行する前に、setLogger を呼び出して、これらの操作から出力をキャプチャすることを確認します。

X-Ray デーモンのアドレス

X-Ray デーモンが、127.0.0.1:2000 以外のポートまたはホスト上でリッスンする場合は、X-Ray SDK for Node.js を使用して、トレースデータを別のアドレスに送信することができます。

```
AWSXRay.setDaemonAddress('host:port');
```

ホストは、名前または IPv4 アドレス で指定できます。

Example app.js - デーモンのアドレス

```
var AWSXRay = require('aws-xray-sdk');
AWSXRay.setDaemonAddress('daemonhost:8082');
```

TCP および UDP の別のポートでリッスンするようデーモンを設定する場合、両方ともデーモンアドレスの設定で指定できます。

Example app.js - 別々のポートのデーモンのアドレス

```
var AWSXRay = require('aws-xray-sdk');
AWSXRay.setDaemonAddress('tcp:daemonhost:8082 udp:daemonhost:8083');
```

または、AWS_XRAY_DAEMON_ADDRESS 環境変数を使用して、デーモンアドレスを設定することもできます。

環境変数

環境変数を使用して、X-Ray SDK for Node.js を設定できます。SDK は次の変数をサポートしています。

- AWS_XRAY_CONTEXT_MISSING – 計測されたコードが、セグメントが開いていないときにデータを記録しようとした場合に例外をスローするには、RUNTIME_ERROR に設定します。

  有効な値

  - RUNTIME_ERROR— ランタイム例外をスローします。
  - LOG_ERROR – エラーをログ記録して続行します (デフォルト)。
  - IGNORE_ERROR – エラーを無視して続行します。

  オープン状態のリクエストがない場合、または新しいスレッドを発生させるコードで、スタートアップコードに実装されたクライアントを使用しようとした場合に発生する可能性がある、セグメントまたはサブセグメントの欠落に関連するエラー。
- AWS_XRAY_DAEMON_ADDRESS – X-Ray デーモンリスナーのホストとポートを設定します。デフォルトでは、SDK はトレースデータ (UDP) とサンプリング (TCP) の両方に127.0.0.1:2000を使

用します。この変数は、デーモンを次のように構成している場合に使用します。別のポートでリッスンするまたは、別のホストで実行されている場合。

[形式]

- 同じポート – *address*:*port*
- 異なるポート – tcp:*address*:*port* udp:*address*:*port*

- AWS_XRAY_DEBUG_MODE— debugレベルでコンソールにログを出力するように SDK を設定するには、TRUEに設定します。
- AWS_XRAY_LOG_LEVEL – デフォルトロガーのログレベルを設定します。有効な値は、debug、info、warn、error、silent です。この値は、AWS_XRAY_DEBUG_MODE が TRUE に設定されている場合、無視されます。
- AWS_XRAY_TRACING_NAME – SDK がセグメントに使用するサービス名を設定します。Express ミドルウェアを設定したセグメント名を上書きします。

X-Ray SDK for Node.js を使用して受信リクエストをトレースします。

X-Ray SDK for Node.js を使用して、Express および Restify アプリケーションが Amazon EC2 の EC2 インスタンス AWS Elastic Beanstalk、または Amazon ECS で処理する受信 HTTP リクエストをトレースできます。 Amazon EC2

X-Ray SDK for Node.js は Express フレームワークおよび Restify フレームワークを使用するアプリケーションのミドルウェアを提供します。X-Ray ミドルウェアをアプリケーションに追加すると、X-Ray SDK for Node.js によってサンプリングされた各リクエストのセグメントが作成されます。このセグメントには、時間、メソッド、HTTP リクエストの処理などが含まれます。追加の計測により、このセグメントでサブセグメントが作成されます。

> **Note**
>
> AWS Lambda 関数の場合、Lambda はサンプリングされたリクエストごとにセグメントを作成します。詳細については、「AWS Lambda および AWS X-Ray」を参照してください。

各セグメントには、サービスマップ内のアプリケーションを識別する名前があります。セグメントの名前は静的に指定することも、受信リクエストのホストヘッダーに基づいて動的に名前を付けるように SDK を設定することもできます。動的ネーミングでは、リクエスト内のドメイン名に基づいてトレースをグループ化でき、名前が予想されるパターンと一致しない場合 (たとえば、ホストヘッダーが偽造されている場合) 、デフォルト名を適用できます。

> **転送されたリクエスト**
>
> ロードバランサーまたは他の仲介者がアプリケーションにリクエストを転送する場合、X-Ray は、クライアントの IP をIP パケットの送信元 IP からではなく、リクエストのX-Forwarded-Forヘッダーから取得します。転送されたリクエストについて記録されたクライアント IP は偽造される可能性があるため、信頼されるべきではありません。

リクエストが転送されると、それを示す追加フィールドが SDK によってセグメントに設定されます。セグメントのフィールド `x_forwarded_for` が `true` に設定されている場合、クライアント IP が HTTP リクエストの `X-Forwarded-For` ヘッダーから取得されます。

メッセージハンドラーは、次の情報が含まれる `http` ブロックを使用して、各受信リクエスト用にセグメントを作成します。

- HTTP メソッド – GET、POST、PUT、DELETE、その他。
- クライアントアドレス – リクエストを送信するクライアントの IP アドレス。
- レスポンスコード – 完了したリクエストの HTTP レスポンスコード。
- タイミング – 開始時間 (リクエストが受信された時間) および終了時間 (レスポンスが送信された時間)。
- ユーザーエージェント — リクエストからの`user-agent`
- コンテンツの長さ — レスポンスからの`content-length`

セクション

Express を使用した受信リクエストのトレース

Express ミドルウェアを使用するには、SDK クライアントを初期化して、ルートを定義する前に `express.openSegment` 関数によって返されたミドルウェアを使用します。

Example app.js - Express

```
var app = express();
```

```
var AWSXRay = require('aws-xray-sdk');
app.use(AWSXRay.express.openSegment('MyApp'));

app.get('/', function (req, res) {
  res.render('index');
});

app.use(AWSXRay.express.closeSegment());
```

ルートを定義した後、express.closeSegment の出力を図のように使用して X-Ray SDK for Node.js によって返されたエラーを処理します。

Restify を使用した受信リクエストのトレース

Restify ミドルウェアを使用するには、SDK クライアントを初期化し、enable を実行します。Restify サーバーおよびセグメント名を渡します。

Example app.js - Restify

```
var AWSXRay = require('aws-xray-sdk');
var AWSXRayRestify = require('aws-xray-sdk-restify');

var restify = require('restify');
var server = restify.createServer();
AWSXRayRestify.enable(server, 'MyApp'));

server.get('/', function (req, res) {
  res.render('index');
});
```

セグメント命名ルールの設定

AWS X-Ray は、サービス名を使用してアプリケーションを識別し、アプリケーションが使用する他のアプリケーション、データベース、外部 APIs、および AWS リソースと区別します。X-Ray SDK が受信リクエストのセグメントを生成すると、アプリケーションのサービス名がセグメントの名前フィールドに記録されます。

X-Ray SDK では、HTTP リクエストヘッダーのホスト名の後にセグメントの名前を指定できます。ただし、このヘッダーは偽造され、サービスマップに予期しないノードが発生する可能性があります。偽造されたホストヘッダーを持つリクエストによって SDK がセグメントの名前を間違えないようにするには、受信リクエストのデフォルト名を指定する必要があります。

アプリケーションが複数のドメインのリクエストを処理する場合、動的ネーミングストラテジーを使用してセグメント名にこれを反映するように SDK を設定できます。動的ネーミングストラテジーにより、SDK は予想されるパターンに一致するリクエストにホスト名を使用し、そうでないリクエストにデフォルト名を適用できます。

たとえば、3 つのサブドメイン (`www.example.com`,`api.example.com`,および`static.example.com`) に対してリクエストを処理する単一のアプリケーションがあるとします。動的ネーミングストラテジーをパターン `*.example.com` で使用して、異なる名前を持つ各サブドメインのセグメントを識別することができます。結果的にはサービスマップ上に 3 つのサービスノードを作成することになります。アプリケーションがパターンと一致しないホスト名のリクエストを受信すると、指定したフォールバック名を持つ 4 番目のノードがサービスマップに表示されます。

すべてのリクエストセグメントに対して同じ名前を使用するには、前のセクションで示すとおり、ミドルウェアを初期化するときに、アプリケーションの名前を指定します。

> **Note**
>
> コードで定義したデフォルトのサービス名は、`AWS_XRAY_TRACING_NAME` 環境変数で上書きできます。

動的な命名戦略は、ホスト名と一致するようパターンを定義し、HTTP リクエストのホスト名がパターンと一致しない場合はデフォルトの名前を使用します。動的にセグメントに命名するには、`AWSXRay.middleware.enableDynamicNaming` を使用します。

Example app.js - 動的セグメントの名前

リクエストのホスト名がパターン `*.example.com` と一致する場合は、そのホスト名を使用します。それ以外の場合は、`MyApp` を使用します。

```
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());
```

X-Ray AWS SDK for Node.js を使用した SDK 呼び出しのトレース

アプリケーションが を呼び出し AWS のサービス てデータの保存、キューへの書き込み、または通知の送信を行う場合、X-Ray SDK for Node.js はサブセグメント の呼び出しダウンストリームを追跡します。トレースされた AWS のサービス、およびそれらのサービス (Amazon S3 バケットや Amazon SQS キューなど) 内でアクセスするリソースは、X-Ray コンソールのトレースマップにダウンストリームノードとして表示されます。

AWS SDK for JavaScript V2 または VAWS SDK for JavaScript V3 を介して作成する AWS SDK クライアントを測定します。 AWS SDK バージョンごとに、 AWS SDK クライアントを計測するためのさまざまな方法が用意されています。

> **Note**
>
> 現在、 AWS X-Ray SDK for Node.js は、V2 クライアントの計測と比較して、 AWS SDK for JavaScript V3 V2クライアントの計測時に少ないセグメント情報を返します。例えば、DynamoDB への呼び出しを表すサブセグメントはテーブル名を返しません。トレースでこのセグメント情報が必要な場合は、 AWS SDK for JavaScript V2 の使用を検討してください。

AWS SDK for JavaScript V2

への呼び出しで aws-sdk require ステートメントをラップすることで、すべての AWS SDK V2 クライアントを計測できますAWSXRay.captureAWS。

Example app.js - AWS SDK 計測

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

個々のクライアントを計測するには、 AWS SDK クライアントを への呼び出しでラップしますAWSXRay.captureAWSClient。たとえば、AmazonDynamoDB クライアントを計測するには:

Example app.js - DynamoDB クライアント計測

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

```
const ddb = AWSXRay.captureAWSClient(new AWS.DynamoDB());
```

> **Warning**
>
> captureAWS と captureAWSClient との両方を使用しないでください。これにより、サブセグメントが重複します。

ECMAScript モジュール (ESM) TypeScriptで を使用してJavaScriptコードをロードする場合は、次の例を使用してライブラリをインポートします。

Example app.js - AWS SDK インストルメンテーション

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

ESM ですべての AWS クライアントを計測するには、次のコードを使用します。

Example app.js - AWS SDK インストルメンテーション

```
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();
```

すべてのサービスにおいて、X-Ray コンソールでコールされた API の名前を確認できます。サービスのサブセットの場合、X-Ray SDK はセグメントに情報を追加して、サービスマップでより細かく指定します。

たとえば、実装された DynamoDB クライアントでコールすると、SDK はテーブルをターゲットとするコールのセグメントにテーブル名を追加します。コンソールで、各テーブルはサービスマップ内に個別のノードとして表示され、テーブルをターゲットにしないコール用の汎用の DynamoDB ノードが表示されます。

Example 項目を保存するための DynamoDB に対するコールのサブセグメント

```
{
  "id": "24756640c0d0978a",
  "start_time": 1.480305974194E9,
  "end_time": 1.4803059742E9,
  "name": "DynamoDB",
```

```
  "namespace": "aws",
  "http": {
    "response": {
      "content_length": 60,
      "status": 200
    }
  },
  "aws": {
    "table_name": "scorekeep-user",
    "operation": "UpdateItem",
    "request_id": "UBQNSO5AEM8T4FDA4RQDEB94OVTDRVV4K4HIRGVJF66Q9ASUAAJG",
  }
}
```

名前付きリソースにアクセスしたとき、次のサービスをコールすると、サービスマップに追加のノードが作成されます。特定のリソースをターゲットとしないコールでは、サービスの汎用ノードが作成されます。

- Amazon DynamoDB – テーブル名
- Amazon Simple Storage Service – バケットとキー名
- Amazon Simple Queue Service – キュー名

AWS SDK for JavaScript V3

AWS SDK for JavaScript V3 はモジュール式であるため、コードは必要なモジュールのみをロードします。このため、V3 は `captureAWS`メソッドをサポートしていないため、すべての AWS SDK クライアントを計測することはできません。

ECMAScript Modules (ESM) TypeScript で を使用してコードをロード JavaScriptする場合は、次の例を使用してライブラリをインポートできます。

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

`AWSXRay.captureAWSv3Client` メソッドを使用して各 AWS SDK クライアントを計測します。たとえば、`AmazonDynamoDB` クライアントを計測するには:

Example app.js - JavaScript V3 用 SDK を使用した DynamoDB クライアント計測

```
    const AWSXRay = require('aws-xray-sdk');
```

```
    const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
...
    const ddb = AWSXRay.captureAWSv3Client(new DynamoDBClient({ region:
 "region" }));
```

AWS SDK for JavaScript V3 を使用する場合、テーブル名、バケット名、キー名、キュー名などのメタデータは現在返されないため、トレースマップには、 AWS SDK for JavaScript V2 を使用して AWS SDK クライアントを計測する場合と同様に、名前付きリソースごとに個別のノードは含まれません。

Example AWS SDK for JavaScript V3 を使用する場合の、項目を保存するための DynamoDB への呼び出しのサブセグメント

```
{
  "id": "24756640c0d0978a",
  "start_time": 1.480305974194E9,
  "end_time": 1.4803059742E9,
  "name": "DynamoDB",
  "namespace": "aws",
  "http": {
    "response": {
      "content_length": 60,
      "status": 200
    }
  },
  "aws": {
    "operation": "UpdateItem",
    "request_id": "UBQNSO5AEM8T4FDA4RQDEB94OVTDRVV4K4HIRGVJF66Q9ASUAAJG",
  }
}
```

X-Ray SDK for Node.js を使用してダウンストリーム HTTP ウェブサービスの呼び出しをトレースする

アプリケーションがマイクロサービスまたはパブリック HTTP API に呼び出しを実行する場合に、X-Ray SDK for Node.js クライアントを使用してこれらの呼び出しを計測し、API をダウンストリームサービスとしてサービスグラフに追加できます。

`http` または `https` クライアントを X-Ray SDK for Node.js の `captureHTTPs` メソッドに渡して、送信呼び出しをトレースします。

> **Note**
>
> Axios や Superagent などのサードパーティー製の HTTP リクエストライブラリを使用する呼び出しは captureHTTPsGlobal() API を通じてサポートされ、ネイティブ http モジュールを使用する場合でもトレースされます。

Example app.js - HTTP クライアント

```
var AWSXRay = require('aws-xray-sdk');
var http = AWSXRay.captureHTTPs(require('http'));
```

すべての HTTP クライアントのトレースを有効にするには、http をロードする前に captureHTTPsGlobal を呼び出します。

Example app.js - HTTP クライアント (グローバル)

```
var AWSXRay = require('aws-xray-sdk');
AWSXRay.captureHTTPsGlobal(require('http'));
var http = require('http');
```

ダウンストリームウェブ API に対する呼び出しを計測すると、X-Ray SDK for Node.js は HTTP リクエストおよびレスポンスに関する情報を含むセグメントを記録します。X-Ray はサブセグメントを使用してリモート API の推測セグメントを生成します。

Example ダウンストリーム HTTP 呼び出しのサブセグメント

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

```
    }
  }
}
```

Example ダウンストリーム HTTP 呼び出しの推定セグメント

```
{
  "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
}
```

X-Ray SDK for Node.js を使用して SQL クエリをトレースします。

SQL クライアントを対応する X-Ray SDK for Node.js クライアントメソッドでラップすることにより、SQL データベースクエリを計測します。

- 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);
```

計測済みクライアントを使用して SQL クエリを作成すると、&X-Ray-nodejssdk; は、サブセグメントに接続およびクエリに関する情報を記録します。

SQL サブセグメントに追加データを含める

許可リストに登録された SQL フィールドにマップされている限り、SQL クエリ用に生成されたサブセグメントに情報を追加できます。たとえば、サニタイズされた SQL クエリ文字列をサブセグメントに記録するには、サブセグメントの SQL オブジェクトに直接追加できます。

Example サブセグメントへの SQL の割り当て

```
    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;
}
```

許可リストに登録されている SQL フィールドの完全なリストについては、「」の「SQL クエリ」セクションを参照してくださいX-Ray セグメントドキュメント。

X-Ray SDK for Node.js を使用したカスタムサブセグメントの生成

サブセグメントはリクエストを処理するために行われた作業の詳細を含んだトレースのセグメントを拡張します。計測済みクライアント内で呼び出しを行うたびに、X-Ray SDK によってサブセグメントに生成された情報が記録されます。追加のサブセグメントを作成して、他のサブセグメントをグループ化したり、コードセクションのパフォーマンスを測定したり、注釈とメタデータを記録したりできます。

カスタム Express サブセグメント

ダウンストリームサービス呼び出しを行う関数用のカスタムセグメントを作成するには、`captureAsyncFunc` 関数を使用します。

Example app.js - カスタムサブセグメント 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();
};
```

この例では、アプリケーションにより、sendRequest 関数への呼び出すために send という名前のカスタムサブセグメントが作成されます。captureAsyncFunc は、非同期呼び出しが完了したときにコールバック関数内で閉じる必要があるサブセグメントを渡します。

同期関数の場合は、captureFunc 関数を使用できます。これにより、関数ブロックの実行が終了するとサブセグメントが自動的に閉じられます。

セグメントまたは別のサブセグメント内にサブセグメントを作成する場合、X-Ray SDK for Node.js によってその ID が生成され、開始時刻と終了時刻が記録されます。

Example サブセグメントとメタデータ

```
"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"
    }
  },
```

カスタム Lambda サブセグメント

SDK は、Lambda で実行中であることを検出したときに、プレースホルダファサードセグメントを自動的に作成するように設定されています。X-Ray トレースマップ上に単一のAWS::Lambda::Functionノードを作成する基本的なサブセグメントを作成するには、 を呼び出し、ファサードセグメントを再利用します。新しい ID を使用して新しいセグメントを手動で作成すると (トレース ID、親 ID、サンプリングデシジョンを共有しているときに)、新しいセグメントを送信できるようになります。

Example app.js - 手動カスタムサブセグメント

```
const segment = AWSXRay.getSegment(); //returns the facade segment
const subsegment = segment.addNewSubsegment('subseg');
...
subsegment.close();
//the segment is closed by the SDK automatically
```

X-Ray SDK for Node.js を使用してセグメントに注釈とメタデータを追加する

注釈とメタデータを使用して、リクエスト、環境、またはアプリケーションに関する追加情報を記録できます。X-Ray SDK が作成するセグメントまたは作成するカスタムサブセグメントに、注釈およびメタデータを追加できます。

注釈は文字列、数値、またはブール値を使用したキーと値のペアです。注釈は、フィルタ式用にインデックス付けされます。注釈を使用して、コンソールでトレースをグループ化するため、または`GetTraceSummaries` API を呼び出すときに使用するデータを記録します。

メタデータは、オブジェクトとリストを含む、任意のタイプの値を持つことができるキーバリューのペアですが、フィルタ式に使用するためにインデックスは作成されません。メタデータを使用してトレースに保存する追加のデータを記録しますが、検索で使用する必要はありません。

注釈とメタデータに加えて、セグメントにユーザー ID 文字列を記録することもできます。ユーザー ID はセグメントの個別のフィールドに記録され、検索用にインデックスが作成されます。

セクション

- X-Ray SDK for Node.js を使用して注釈を記録
- X-Ray SDK for Node.js を使用したメタデータの記録
- X-Ray SDK for Node.js を使用したユーザー ID の記録

X-Ray SDK for Node.js を使用して注釈を記録

注釈を使用して、検索用にインデックスを作成するセグメントまたはサブセグメントに情報を記録します。

注釈の要件

- キー – X-Ray 注釈のキーには、最大 500 文字の英数字を使用できます。アンダースコア記号 (_) 以外のスペースや記号は使用できません。
- 値 - X-Ray 注釈の値は、最大 1,000 文字の Unicode 文字を持つことができます。
- 注釈の数 — トレースごとに最大 50 個の注釈を使用できます。

注釈を記録するには

1. 現在のセグメントまたはサブセグメントの参照を取得します。

```
var AWSXRay = require('aws-xray-sdk');
...
var document = AWSXRay.getSegment();
```

2. 文字列キー、および、ブール値、数値、文字列値を使用して addAnnotation を呼び出します。

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

SDK は、セグメントドキュメントの annotations オブジェクトにキーと値のペアとして、注釈を記録します。同じキーで addAnnotation を 2 回呼び出すと、同じセグメントまたはサブセグメントに以前記録された値が上書きされます。

特定の値を持つ注釈のあるトレースを見つけるには、annotations.*key*フィルタ式の キーワードを使用します。

Example app.js - 注釈

```
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) {
...
```

X-Ray SDK for Node.js を使用したメタデータの記録

メタデータを使用して、検索用にインデックスを作成する必要のないセグメントまたはサブセグメントに情報を記録します。メタデータ値は、文字列、数値、ブール値、または JSON オブジェクトや JSON 配列にシリアル化できるその他の任意のオブジェクトになります。

メタデータを記録するには

1. 現在のセグメントまたはサブセグメントの参照を取得します。

```
var AWSXRay = require('aws-xray-sdk');
...
var document = AWSXRay.getSegment();
```

2. 文字列キー、ブール値、数値、文字列値、オブジェクト値、文字列名前空間を使用して addMetadata を呼び出します。

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

   または

   キーと値だけを使用して addMetadata を呼び出します。

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

名前空間を指定しない場合、SDK は default を使用します。同じキーで addMetadata を 2 回呼び出すと、同じセグメントまたはサブセグメントに以前記録された値が上書きされます。

X-Ray SDK for Node.js を使用したユーザー ID の記録

リクエストセグメントにユーザー ID を記録して、リクエストを送信したユーザーを識別します。Lambda 環境のセグメントはイミュータブルであるため、このオペレーションは AWS Lambda 関数と互換性がありません。setUser の呼び出しはセグメントにのみ適用でき、サブセグメントには適用できません。

ユーザー ID を記録するには

1. 現在のセグメントまたはサブセグメントの参照を取得します。

```
var AWSXRay = require('aws-xray-sdk');
```

```
...
var document = AWSXRay.getSegment();
```

2. リクエストを送信したユーザーの文字列 ID を使用して `setUser()` を呼び出します。

```
var user = 'john123';

AWSXRay.getSegment().setUser(user);
```

`setUser` を呼び出し、Express アプリケーションがリクエストの処理を開始するとすぐに、ユーザー ID を記録できます。ユーザー ID を設定するためだけにセグメントを使用する場合、呼び出しを 1 行で連鎖させることができます。

Example app.js - ユーザー ID

```
var AWS = require('aws-sdk');
var AWSXRay = require('aws-xray-sdk');
var uuidv4 = require('uuid/v4');
var ddb = AWSXRay.captureAWSClient(new AWS.DynamoDB());
...
    app.post('/signup', function(req, res) {
    var userId = uuidv4();
    var item = {
        'userId': {'S': userId},
        'email': {'S': req.body.email},
        'name': {'S': req.body.name}
    };

    var seg = AWSXRay.getSegment().setUser(userId);

    ddb.putItem({
      'TableName': ddbTable,
      'Item': item,
      'Expected': { email: { Exists: false } }
  }, function(err, data) {
...
```

ユーザー ID のトレースを見つけるには、userフィルタ式で、 キーワードを使用します。

を使用してアプリケーションを計測する Python

X-Ray にトレースを送信するようにPythonアプリケーションを計測するには、次の 2 つの方法があります。

- AWS Distro for OpenTelemetry Python – AWS Distro for OpenTelemetry Collector を介して相関メトリクスとトレースを Amazon CloudWatch、 AWS X-Ray、Amazon OpenSearch Service などの複数の AWS モニタリングソリューションに送信するための一連のオープンソースライブラリを提供するディス AWS トリビューション。
- AWS X-Ray SDK for Python – X-Ray デーモン を介してトレースを生成して X-Ray に送信するためのライブラリのセット。

詳細については、「AWS Distro for OpenTelemetry と X-Ray SDKs の選択」を参照してください。

AWS のディストリビューション OpenTelemetry Python

AWS Distro for OpenTelemetry (ADOT) を使用するとPython、アプリケーションを一度インストゥルメントすれば、相関関係のあるメトリクスとトレースを Amazon CloudWatch、 AWS X-Ray、Amazon Service AWS などの複数のモニタリングソリューションに送信できます。OpenSearch X-Ray を ADOT で使用するには、2 つのコンポーネントが必要です。1 つは X-Ray で使用できる OpenTelemetry SDK で、もう 1 つは X-Ray で使用可能な OpenTelemetryCollector AWS 用ディストリビューションです。ADOT Python には自動インストルメンテーションサポートが含まれているため、アプリケーションはコードを変更せずにトレースを送信できます。

はじめに、ディストリビューションのドキュメントを参照してください。AWS OpenTelemetry Python

OpenTelemetry with AWS X-Ray OpenTelemetry や other 用の AWS Distro の使用について詳しくは、「Distro」を AWS のサービス、AWS ドキュメンテーションについては「Distro」を参照してください。AWS OpenTelemetry

言語のサポートと使用方法について詳しくは、「オブザーバビリティオン」を参照してください AWS 。 GitHub

AWS X-Ray 用の SDK Python

X-Ray SDK for Python は、トレースデータを生成して X-Ray デーモンに送信するためのクラスとメソッドを提供する Python Web アプリケーション用のライブラリです。トレースデータには、

アプリケーションが処理した受信 HTTP リクエストに関する情報と、アプリケーションが AWS SDK、HTTP クライアント、または SQL データベースコネクタを使用してダウンストリームサービスに対して行う呼び出しに関する情報が含まれます。セグメントを手動で作成し、注釈およびメタデータにデバッグ情報を追加することもできます。

SDK は `pip` を使用してダウンロードできます。

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

> **Note**
>
> X-Ray SDK for Python は、オープンソースプロジェクトです。github.com/aws/ でプロジェクトをフォローし、課題やプルリクエストを送信できます GitHub。aws-xray-sdk-python

Django または Flask を使用する場合は、アプリケーションに SDK ミドルウェアを追加し、受信リクエストをトレースします。ミドルウェアでは、トレース対象リクエストごとに「セグメント」を作成し、レスポンスが送信されるとセグメントを完了します。セグメントが開いている間、SDK クライアントのメソッドを使用してセグメントに情報を追加し、サブセグメントを作成してダウンストリーム呼び出しをトレースできます。また、SDK では、セグメントが開いている間にアプリケーションがスローする例外を自動的に記録します。他のアプリケーションの場合、手動でセグメントを作成することができます。

測定されたアプリケーションまたはサービスから呼び出された Lambda 関数に対して、Lambda はトレースヘッダー を読み込み、サンプリングされたリクエストを自動的にトレースします。その他の関数については、Lambda の設定 から受信リクエストのサンプリングとトレースを行うことができます。いずれの場合も、Lambda はセグメントを作成し、X-Ray SDK に提供します。

> **Note**
>
> Lambda では、X-Ray SDK はオプションです。関数でこれを使用しない場合、サービスマップには Lambda サービスのノードと Lambda 関数ごとに 1 つのノードが含まれます。SDK を追加することで、関数コードをインストルメントして、Lambda で記録された関数セグメントにサブセグメントを追加することができます。詳細については、「AWS Lambda および AWS X-Ray」を参照してください。

Lambda ワーカー Python でインストゥルメントされた関数の例については、を参照してください。

次に、X-Ray SDK for Python を使用してダウンストリーム呼び出しを実装するには、アプリケーションが使用するライブラリにパッチを適用します。SDK は次のライブラリをサポートしています。

サポートされているライブラリ

- botocore, boto3 — インストゥルメントクライアント。 AWS SDK for Python (Boto)
- pynamodb – 測定された Amazon DynamoDB クライアントの PynamoDB のバージョン。
- aiobotocore、aioboto3 - 測定された asyncio統合バージョンの SDK for Python クライアント。
- requests、aiohttp - 測定された高レベルの HTTP クライアント。
- httplib、 http.client - 測定された低レベルの HTTP クライアントおよびそれらを使用する高レベルのライブラリ。
- sqlite3 - SQLite クライアントを測定します。
- mysql-connector-python - MySQL クライアントを測定します。
- pg8000 - Pure-Python PostgreSQL インターフェイスを測定します。
- psycopg2 - PostgreSQL データベースアダプターを測定します。
- pymongo - MongoDB クライアントを測定します。
- pymysql— PyMy MySQL および MariaDB 用の SQL ベースのクライアントをインストゥルメントします。

アプリケーションが SQL データベース AWS、またはその他の HTTP サービスを呼び出すたびに、SDK はその呼び出しに関する情報をサブセグメントに記録します。 AWS のサービス また、サービス内でアクセスするリソースはトレースマップに下流ノードとして表示されるため、個々の接続のエラーやスロットリングの問題を特定するのに役立ちます。

SDK を入手したら、レコーダーとミドルウェアを設定して、その動作をカスタマイズします。プラグインを追加して、アプリケーションを実行しているコンピューティングリソースに関するデータを記録したり、サンプリングルールを定義することでサンプリングの動作のカスタマイズしたり、アプリケーションログに SDK からの情報をより多くあるいは少なく表示するようにログレベルを設定できます。

アプリケーションが注釈やメタデータで行うリクエストや作業に関する追加情報を記録します。注釈は、フィルタ式で使用するためにインデックス化されたシンプルなキーと値のペアで、特定のデータが含まれているトレースを検索できます。メタデータのエントリは制約が緩やかで、JSON にシリアル化できるオブジェクトと配列全体を記録できます。

> **注釈とメタデータ**
>
> 注釈およびメタデータとは、X-Ray SDK を使用してセグメントに追加する任意のテキストです。注釈は、フィルタ式用にインデックス付けされます。メタデータはインデックス化されませんが、X-Ray コンソールまたは API を使用して raw セグメントで表示できます。X-Ray への読み取りアクセスを許可した人は誰でも、このデータを表示できます。

コードに多数の計測されたクライアントがある場合、単一のリクエストセグメントには計測されたクライアントで行われた呼び出しごとに 1 個の多数のサブセグメントを含めることができます。カスタムサブセグメントで、クライアント呼び出しをラップすることで、サブセグメントを整理してグループできます。関数全体またはコードの任意のセクションに対して、カスタムサブセグメントを作成できます。親セグメントのすべてを書き込むのではなく、サブセグメントにメタデータと注釈を記録することができます。

SDK のクラスとメソッドのリファレンスドキュメントについては、SDK AWS X-Ray for API リファレンスをご覧ください。Python

要件

X-Ray SDK for Python では、次の言語とライブラリのバージョンがサポートされています。

- Python – 2.7、3.4 以降
- Django – 1.10 以降
- Flask – 0.10 以降
- aiohttp – 2.3.0 以降
- AWS SDK for Python (Boto) - 1.4.0 以降
- botocore – 1.5.0 以降
- 列挙型 — 0.4.7 以降、Pythonバージョン 3.4.0 以前の場合
- jsonpickle – 1.0.0 以降
- setuptools – 40.6.3 以降
- wrapt – 1.11.0 以降

依存関係管理

X-Ray SDK for Python は、`pip`から入手できます。

- パッケージ – `aws-xray-sdk`

1. SDK を依存関係として `requirements.txt` ファイルに追加します。

Example requirements.txt

```
aws-xray-sdk==2.4.2
boto3==1.4.4
botocore==1.5.55
Django==1.11.3
```

Elastic Beanstalk を使用してアプリケーションをデプロイする場合、Elastic Beanstalk は `requirements.txt` のパッケージをすべて自動的にインストールします。

X-Ray SDK for Python の設定

X-Ray SDK for Python には、グローバルレコーダーを提供する `xray_recorder` というクラスがあります。グローバルレコーダーを設定して、受信 HTTP コールのセグメントを作成するミドルウェアをカスタマイズできます。

セクション

- サービスプラグイン
- サンプリングルール
- ログ記録
- コード内のレコーダー設定
- Django でのレコーダー設定
- 環境変数

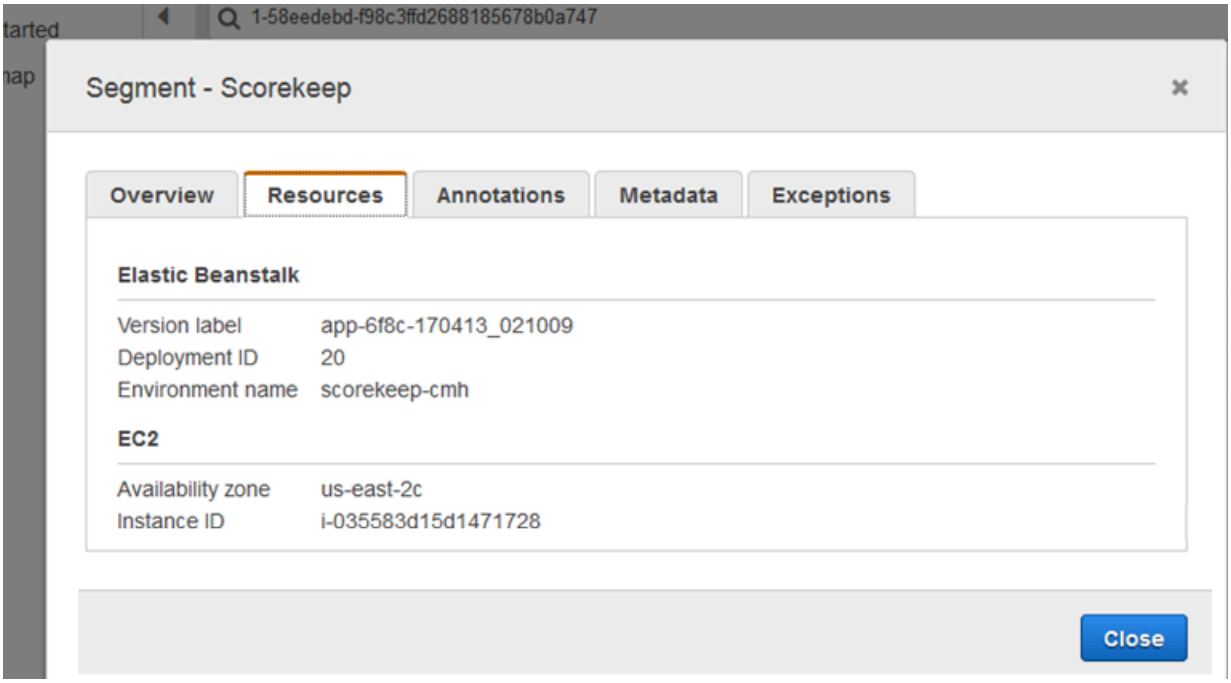
サービスプラグイン

`plugins`を使用して、アプリケーションをホストしているサービスに関する情報を記録します。

プラグイン

- Amazon EC2 – は、インスタンス ID、アベイラビリティーゾーン、および CloudWatch Logs グループ`EC2Plugin`を追加します。
- ElasticBeanstalk– `ElasticBeanstalkPlugin`は、環境名、バージョンラベル、およびデプロイ ID を追加します。

- Amazon ECS —ECSPluginは、コンテナ ID を追加します。



プラグインを使用するには、configure で xray_recorder を呼び出します。1 はタプルとして渡されるため、単一のプラグインを指定する場合は必ず末尾に 2 を付けてください。

```
from aws_xray_sdk.core import xray_recorder
from aws_xray_sdk.core import patch_all

xray_recorder.configure(service='My app')
plugins = ('ElasticBeanstalkPlugin', 'EC2Plugin')
xray_recorder.configure(plugins=plugins)
patch_all()
```

> **Note**
>
> plugins はタプルとして渡されるため、単一のプラグインを指定する場合は必ず末尾に , を付けてください。例えば、plugins = ('EC2Plugin',)

また、コードで設定した値よりも優先される環境変数を使用して、レコーダーを設定することもできます。

ダウンストリームコールを記録するには、パッチライブラリの前にプラグインを設定します。

また、SDK はプラグインの設定を利用して、セグメントの `origin` フィールドを設定します。これは、アプリケーションを実行する AWS リソースのタイプを示します。複数のプラグインを使用する場合、SDK は次の解決順序を使用してオリジンを決定します： ElasticBeanstalk > EKS > ECS > EC2。

サンプリングルール

SDK は X-Ray コンソールで定義したサンプリングルールを使用し、記録するリクエストを決定します。デフォルトルールでは、最初のリクエストを毎秒トレースし、X-Ray にトレースを送信するすべてのサービスで追加のリクエストの 5% をトレースします。X-Ray コンソールに追加のルールを作成するをクリックして、各アプリケーションで記録されるデータ量をカスタマイズします。

SDK は、定義された順序でカスタムルールを適用します。リクエストが複数のカスタムルールと一致する場合、SDK は最初のルールのみを適用します。

> **Note**
>
> SDK が X-Ray に到達してサンプリングルールを取得できない場合、1 秒ごとに受信された最初のリクエストのデフォルトのローカルルールに戻り、ホストあたりの追加リクエストの 5% に戻ります。これは、ホストがサンプリング API を呼び出す権限を持っていない場合や、SDK によって行われる API 呼び出しの TCP プロキシとして機能する X-Ray デーモンに接続できない場合に発生します。

JSON ドキュメントからサンプリングルールをロードするように SDK を設定することもできます。SDK は、X-Ray サンプリングが利用できない場合のバックアップとしてローカルルールを使用することも、ローカルルールを排他的に使用することもできます。

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
  }
}
```

この例では、1 つのカスタムルールとデフォルトルールを定義します。カスタムルールでは、5 パーセントのサンプリングレートが適用され、/api/move/以下のパスに対してトレースするリクエストの最小数はありません。デフォルトのルールでは、1秒ごとの最初のリクエストおよび追加リクエストの 10 パーセントをトレースします。

ルールをローカルで定義することの欠点は、固定ターゲットが X-Ray サービスによって管理されるのではなく、レコーダーの各インスタンスによって個別に適用されることです。より多くのホストをデプロイすると、固定レートが乗算され、記録されるデータ量の制御が難しくなります。

では AWS Lambda、サンプリングレートを変更することはできません。関数がインストルメント化されたサービスによって呼び出された場合、そのサービスによってサンプリングされたリクエストを生成した呼び出しは Lambda によって記録されます。アクティブなトレースが有効で、トレースヘッダーが存在しない場合、Lambda はサンプリングを決定します。

バックアップサンプリングルールを設定するには、次の例に示すように、`xray_recorder.configure` を呼び出します。*`rules`* は、ルールの辞書または JSON ファイルの絶対パスサンプリングルールです。

```
xray_recorder.configure(sampling_rules=rules)
```

ローカルルールのみを使用するには、`LocalSampler` でレコーダーを設定します。

```
from aws_xray_sdk.core.sampling.local.sampler import LocalSampler
xray_recorder.configure(sampler=LocalSampler())
```

サンプリングを無効にしてすべての着信リクエストを実装するように、グローバルレコーダーを設定することもできます。

Example main.py - サンプリングを無効にする

```
xray_recorder.configure(sampling=False)
```

ログ記録

SDK は、Python の組み込み logging モジュールを使用し、WARNINGデフォルトのログ記録レベルを使用します。aws_xray_sdk クラスのロガーへの参照を取得し、その上で setLevel を呼び出して、ライブラリとアプリケーションに別のログレベルを設定します。

Example app.py - ログ記録

```
logging.basicConfig(level='WARNING')
logging.getLogger('aws_xray_sdk').setLevel(logging.ERROR)
```

デバッグログを使用して問題を識別します。たとえば、「サブセグメントを手動で生成する」場合にサブセグメントが閉じない問題などです。

コード内のレコーダー設定

追加の設定は、xray_recorder の configure メソッドから利用できます。

- context_missing - 測定されたコードが、セグメントが開いていないときにデータを記録しようとした場合に例外のスローを回避するには、LOG_ERROR に設定します。
- daemon_address – X-Ray デーモンリスナーのホストとポートを設定します。
- service - SDK がセグメントに使用するサービス名を設定します。
- plugins – アプリケーションの AWS リソースに関する情報を記録します。
- sampling – False に設定してサンプリングを無効にします。
- sampling_rules – サンプリングルールを含む JSON ファイルのパスを設定します。

Example main.py - コンテキスト欠落例外を無効にする

```
from aws_xray_sdk.core import xray_recorder

xray_recorder.configure(context_missing='LOG_ERROR')
```

Django でのレコーダー設定

Django フレームワークを使用している場合は、Django settings.py ファイルを使用してグローバルレコーダーのオプションを設定できます。

- AUTO_INSTRUMENT (Django のみ) - 組み込みデータベースおよびテンプレートレンダリング操作のサブセグメントを記録します。

- AWS_XRAY_CONTEXT_MISSING - LOG_ERROR に設定すると、セグメントを開いていないときに測定されたコードがデータを記録しようとしたときに例外が発生するのを防ぐことができます。
- AWS_XRAY_DAEMON_ADDRESS – X-Ray デーモンリスナーのホストとポートを設定します。
- AWS_XRAY_TRACING_NAME - SDK がセグメントに使用するサービス名を設定します。
- PLUGINS – アプリケーションの AWS リソースに関する情報を記録します。
- SAMPLING – False に設定してサンプリングを無効にします。
- SAMPLING_RULES – サンプリングルールを含む JSON ファイルのパスを設定します。

settings.py でレコーダ設定を有効にするには、インストールされているアプリのリストに Django ミドルウェアを追加します。

Example settings.py - インストールされたアプリ

```
INSTALLED_APPS = [
    ...
    'django.contrib.sessions',
    'aws_xray_sdk.ext.django',
]
```

XRAY_RECORDER という名前の dict で利用可能な設定を行います。

Example settings.py - インストールされたアプリ

```
XRAY_RECORDER = {
    'AUTO_INSTRUMENT': True,
    'AWS_XRAY_CONTEXT_MISSING': 'LOG_ERROR',
    'AWS_XRAY_DAEMON_ADDRESS': '127.0.0.1:5000',
    'AWS_XRAY_TRACING_NAME': 'My application',
    'PLUGINS': ('ElasticBeanstalkPlugin', 'EC2Plugin', 'ECSPlugin'),
    'SAMPLING': False,
}
```

環境変数

環境変数を使用して、X-Ray SDK for Python を設定できます。SDK は次の変数をサポートしています。

- AWS_XRAY_TRACING_NAME - SDK がセグメントに使用するサービス名を設定します。プログラムによって設定したサービス名を上書きします。

- AWS_XRAY_SDK_ENABLED – falseに設定されている場合、SDK は無効になります。デフォルトでは、環境変数が false に設定されている場合を除き、SDK は有効です。
  - 無効にすると、グローバルレコーダーによって、デーモンに送信されないダミーセグメントとサブセグメントが自動的に生成され、自動パッチ適用は無効になります。ミドルウェアはグローバルレコーダーのラッパーとして記述されています。ミドルウェアによるセグメントやサブセグメントの生成もすべて、ダミーセグメントおよびダミーサブセグメントになります。
  - AWS_XRAY_SDK_ENABLED の値を設定するには、環境変数を使用するか、aws_xray_sdk ライブラリの global_sdk_config オブジェクトと直接やり取りします。環境変数を設定すると、これらの通信は上書きされます。
- AWS_XRAY_DAEMON_ADDRESS – X-Ray デーモンリスナーのホストとポートを設定します。デフォルトでは、SDK はトレースデータ (UDP) とサンプリング (TCP) の両方に127.0.0.1:2000を使用します。この変数は、デーモンを次のように構成している場合に使用します。別のポートでリッスンするまたは、別のホストで実行されている場合。

  [形式]

  - 同じポート – *address*:*port*
  - 異なるポート – tcp:*address*:*port* udp:*address*:*port*
- AWS_XRAY_CONTEXT_MISSING – 計測されたコードが、セグメントが開いていないときにデータを記録しようとした場合に例外をスローするには、RUNTIME_ERROR に設定します。

  有効な値

  - RUNTIME_ERROR— ランタイム例外をスローします。
  - LOG_ERROR – エラーをログ記録して続行します (デフォルト)。
  - IGNORE_ERROR – エラーを無視して続行します。

  リクエストが開かれていないときに実行されるスタートアップコード、または新しいスレッドを生成するコードで測定されたクライアントを使用しようとしたときに発生する可能性があるセグメントまたはサブセグメントの欠落に関連するエラー。

環境変数は、コードで設定される値を上書きします。

X-Ray SDK for Python ミドルウェアを使用して受信リクエストをトレースします。

ミドルウェアをアプリケーションに追加してセグメント名を設定すると、X-Ray SDK for Python はサンプリングされた要求ごとにセグメントを作成します。このセグメントには、時間、メソッ

ド、HTTP リクエストの処理などが含まれます。追加の計測により、このセグメントでサブセグメントが作成されます。

X-Ray SDK for Python は、受信 HTTP リクエストを測定する次のミドルウェアをサポートしています。

- Django
- Flask
- Bottle

> **Note**
>
> AWS Lambda 関数の場合、Lambda はサンプリングされたリクエストごとにセグメントを作成します。詳細については、「AWS Lambda および AWS X-Ray」を参照してください。

Lambda で測定されているサンプル Python 関数については、「ワーカー」を参照してください。

他のフレームワークのスクリプトまたは Python アプリケーションの場合、セグメントを手動で作成することができます。

各セグメントには、サービスマップ内のアプリケーションを識別する名前があります。セグメントの名前は静的に指定することも、受信リクエストのホストヘッダーに基づいて動的に名前を付けるように SDK を設定することもできます。動的ネーミングでは、リクエスト内のドメイン名に基づいてトレースをグループ化でき、名前が予想されるパターンと一致しない場合 (たとえば、ホストヘッダーが偽造されている場合) 、デフォルト名を適用できます。

> **転送されたリクエスト**
>
> ロードバランサーまたは他の仲介者がアプリケーションにリクエストを転送する場合、X-Ray は、クライアントの IP をIP パケットの送信元 IP からではなく、リクエストのX-Forwarded-Forヘッダーから取得します。転送されたリクエストについて記録されたクライアント IP は偽造される可能性があるため、信頼されるべきではありません。

リクエストが転送されると、それを示す追加フィールドが SDK によってセグメントに設定されます。セグメントのフィールド `x_forwarded_for` が `true` に設定されている場合、クライアント IP が HTTP リクエストの `X-Forwarded-For` ヘッダーから取得されます。

ミドルウェアは、次の情報が含まれる http ブロックを使用して、各受信リクエスト用にセグメントを作成します。

- HTTP メソッド – GET、POST、PUT、DELETE、その他。
- クライアントアドレス – リクエストを送信するクライアントの IP アドレス。
- レスポンスコード – 完了したリクエストの HTTP レスポンスコード。
- タイミング – 開始時間 (リクエストが受信された時間) および終了時間 (レスポンスが送信された時間)。
- ユーザーエージェント — リクエストからのuser-agent
- コンテンツの長さ — レスポンスからのcontent-length

セクション

- アプリケーションにミドルウェアを追加する (Django)
- アプリケーションにミドルウェアを追加する (Flask)
- アプリケーションにミドルウェアを追加する (Bottle)
- Python コードを手動で実装する
- セグメント命名ルールの設定

アプリケーションにミドルウェアを追加する (Django)

ミドルウェアを MIDDLEWARE ファイルの settings.py リストに追加します。X-Ray ミドルウェアは、他のミドルウェアで失敗した要求が確実に記録されるように、settings.py ファイルの最初の行にする必要があります。

Example settings.py - X-Ray SDK for Python ミドルウェア

```
MIDDLEWARE = [
    'aws_xray_sdk.ext.django.middleware.XRayMiddleware',
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware'
]
```

settings.py ファイルの INSTALLED_APPS リストに X-Ray SDK Django アプリを追加します。これにより、アプリの起動時にX-Ray レコーダーを設定できるようになります。

Example settings.py - X-Ray SDK for Python Django アプリ

```
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',
]
```

使用する settings.py ファイルにセグメント名を設定します。

Example settings.py - セグメント名

```
XRAY_RECORDER = {
    'AWS_XRAY_TRACING_NAME': 'My application',
    'PLUGINS': ('EC2Plugin',),
}
```

これは、X-Ray レコーダーに、デフォルトのサンプリングレートで Django アプリケーションが提供する要求をトレースするように指示します。使用する Django 設定ファイルにレコーダーを設定して、カスタムサンプリングルールを適用するか、その他の設定を変更することができます。

> **Note**
>
> plugins はタプルとして渡されるため、単一のプラグインを指定する場合は必ず末尾に , を付けてください。例えば、plugins = ('EC2Plugin',) などです。

アプリケーションにミドルウェアを追加する (Flask)

Flask アプリケーションを計測するには、最初に xray_recorder にセグメント名を設定します。次に、XRayMiddleware 関数を使用して Flask アプリケーションをコードにパッチします。

Example app.py

```
from aws_xray_sdk.core import xray_recorder
```

```
from aws_xray_sdk.ext.flask.middleware import XRayMiddleware

app = Flask(__name__)

xray_recorder.configure(service='My application')
XRayMiddleware(app, xray_recorder)
```

これは、X-Ray レコーダーに、デフォルトのサンプリングレートで Flask アプリケーションが提供する要求をトレースするように指示します。レコーダーをコードに設定して、カスタムサンプリングルールを適用するか、その他の設定を変更することができます。

アプリケーションにミドルウェアを追加する (Bottle)

Bottle アプリケーションを計測するには、最初に `xray_recorder` にセグメント名を設定します。次に、`XRayMiddleware` 関数を使用して Bottle アプリケーションをコードにパッチします。

Example app.py

```
from aws_xray_sdk.core import xray_recorder
from aws_xray_sdk.ext.bottle.middleware import XRayMiddleware

app = Bottle()

xray_recorder.configure(service='fallback_name', dynamic_naming='My application')
app.install(XRayMiddleware(xray_recorder))
```

これは、X-Ray レコーダーに、デフォルトのサンプリングレートで Bottle アプリケーションが提供する要求をトレースするように指示します。レコーダーをコードに設定して、カスタムサンプリングルールを適用するか、その他の設定を変更することができます。

Python コードを手動で実装する

Django または Flask を使用していない場合、手動でセグメントを作成することができます。受信リクエストごとにセグメントを作成したり、パッチが適用された HTTP または AWS SDK クライアントの周囲にセグメントを作成してレコーダーのコンテキストを提供し、サブセグメントを追加したりすることができます。

Example main.py - 手動測定

```
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()
```

セグメント命名ルールの設定

AWS X-Rayはサービス名を使用してアプリケーションを識別し、他のアプリケーション、データベース、外部 API、およびAWSアプリケーションが使用するリソースと区別します。X-Ray SDK が受信リクエストのセグメントを生成すると、アプリケーションのサービス名がセグメントの名前フィールドに記録されます。

X-Ray SDK では、HTTP リクエストヘッダーのホスト名の後にセグメントの名前を指定できます。ただし、このヘッダーは偽造され、サービスマップに予期しないノードが発生する可能性があります。偽造されたホストヘッダーを持つリクエストによって SDK がセグメントの名前を間違えないようにするには、受信リクエストのデフォルト名を指定する必要があります。

アプリケーションが複数のドメインのリクエストを処理する場合、動的ネーミングストラテジーを使用してセグメント名にこれを反映するように SDK を設定できます。動的ネーミングストラテジーにより、SDK は予想されるパターンに一致するリクエストにホスト名を使用し、そうでないリクエストにデフォルト名を適用できます。

たとえば、3 つのサブドメイン (`www.example.com`,`api.example.com`,および`static.example.com`) に対してリクエストを処理する単一のアプリケーションがあるとします。動的ネーミングストラテジーをパターン `*.example.com` で使用して、異なる名前を持つ各サブドメインのセグメントを識別することができます。結果的にはサービスマップ上に 3 つのサービスノードを作成することになります。アプリケーションがパターンと一致しないホスト名のリクエストを受信すると、指定したフォールバック名を持つ 4 番目のノードがサービスマップに表示されます。

すべてのリクエストセグメントに同じ名前を使用するには、前のセクションで示されたように、レコーダーを設定するときにアプリケーションの名前を指定します。

動的命名ルールは、ホスト名と一致するようパターンを定義し、HTTP リクエストのホスト名がパターンと一致しない場合はデフォルトの名前を使用します。Django でセグメントに動的な名前を付けるには、DYNAMIC_NAMING 設定を使用する settings.py ファイルに追加します。

Example settings.py - 動的ネーミング

```
XRAY_RECORDER = {
    'AUTO_INSTRUMENT': True,
    'AWS_XRAY_TRACING_NAME': 'My application',
    'DYNAMIC_NAMING': '*.example.com',
    'PLUGINS': ('ElasticBeanstalkPlugin', 'EC2Plugin')
}
```

パターン内で任意の文字列に一致させるには「*」を、また、任意の 1 文字に一致させるには「?」を使用することができます。Flask の場合は、コードでレコーダーを設定します。

Example main.py - セグメント名

```
from aws_xray_sdk.core import xray_recorder
xray_recorder.configure(service='My application')
xray_recorder.configure(dynamic_naming='*.example.com')
```

> **Note**
>
> コードで定義したデフォルトのサービス名は、AWS_XRAY_TRACING_NAME 環境変数で上書きできます。

ダウンストリームコールを実装するためのライブラリへのパッチ適用

ダウンストリーム呼び出しを測定するには、X-Ray SDK for Python を使用して、アプリケーションが使用するライブラリにパッチを適用します。X-Ray SDK for Python では、以下のライブラリにパッチを適用できます。

サポートされているライブラリ

- botocore、boto3 - 測定された AWS SDK for Python (Boto) クライアント。
- pynamodb – 測定された Amazon DynamoDB クライアントの PynamoDB のバージョン。
- aiobotocore、aioboto3 - 測定された asyncio統合バージョンの SDK for Python クライアント。

- requests、aiohttp - 測定された高レベルの HTTP クライアント。
- httplib、 http.client - 測定された低レベルの HTTP クライアントおよびそれらを使用する高レベルのライブラリ。
- sqlite3 - SQLite クライアントを測定します。
- mysql-connector-python - MySQL クライアントを測定します。
- pg8000 - Pure-Python PostgreSQL インターフェイスを測定します。
- psycopg2 - PostgreSQL データベースアダプターを測定します。
- pymongo - MongoDB クライアントを測定します。
- pymysql - MySQL と MariaDB 用 PyMySQL ベースクライアントを測定します。

パッチ適用されたライブラリを使用すると、X-Ray SDK for Python は呼び出しのサブセグメントが作成され、リクエストとレスポンスの情報を記録します。SDK ミドルウェアまたは AWS Lambda のいずれかから、サブセグメントを作成するためにSDK でセグメントを使用できる必要があります。

> **Note**
>
> SQLAlchemy ORM を使用する場合は、SQLAlchemy のセッションクラスとクエリクラスの SDK のバージョンをインポートして、SQL クエリを追加できます。手順については、Use SQLAlchemy ORM を参照してください。

使用可能なすべてのライブラリにパッチを適用するには、`aws_xray_sdk.core` の `patch_all` 関数を使用します。`httplib` や `urllib` などの一部のライブラリでは、`patch_all(double_patch=True)` を呼び出して二重パッチ適用を有効にすることが必要な場合があります。

Example main.py - サポートされているすべてのライブラリにパッチを適用

```
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()
```

単一のライブラリにパッチを適用するには、ライブラリ名のタプルを使用して patch を呼び出します。これを行うには、単一の要素リストを用意する必要があります。

Example main.py - 特定のライブラリにパッチを適用

```
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**
>
> 場合によっては、ライブラリにパッチを適用するために使用するキーがライブラリ名と一致しない場合があります。一部のキーは、1 つまたは複数のライブラリのエイリアスとして機能します。
>
> ライブラリエイリアス
>
> - `httplib` – `httplib` および `http.client`
> - `mysql` – `mysql-connector-python`

非同期作業のコンテキストのトレース

`asyncio` によって統合されたライブラリの場合や、非同期関数用のサブセグメントを作成する場合は、非同期コンテキストで X-Ray SDK for Python も設定する必要があります。`AsyncContext` クラスをインポートし、そのインスタンスを X-Ray レコーダーに渡します。

> **Note**
>
> ウェブフレームワークサポートライブラリ (例: AIOHTTP) は、`aws_xray_sdk.core.patcher` モジュールで処理することはできません。これらのライブラリは、サポートされているライブラリの `patcher` カタログに表示されません。

Example main.py - aioboto3 にパッチを適用

```
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)
```

X-Ray AWS SDK for Python を使用した SDK 呼び出しのトレース

アプリケーションが を呼び出し AWS のサービス てデータの保存、キューへの書き込み、または通知の送信を行う場合、X-Ray SDK for Python はサブセグメント の呼び出しダウンストリームを追跡します。これらのサービス (Amazon S3 バケットや Amazon SQS キューなど) 内でアクセスするトレースされた AWS のサービス および リソースは、X-Ray コンソールのトレースマップにダウンストリームノードとして表示されます。

X-Ray SDK for Python は、ライブラリ にパッチを適用すると、すべての AWS SDK クライアントを自動的に計測します。 `botocore`個々のクライアントを実装することはできません。

すべてのサービスにおいて、X-Ray コンソールでコールされた API の名前を確認できます。サービスのサブセットの場合、X-Ray SDK はセグメントに情報を追加して、サービスマップでより細かく指定します。

たとえば、実装された DynamoDB クライアントでコールすると、SDK はテーブルをターゲットとするコールのセグメントにテーブル名を追加します。コンソールで、各テーブルはサービスマップ内に個別のノードとして表示され、テーブルをターゲットにしないコール用の汎用の DynamoDB ノードが表示されます。

Example 項目を保存するための DynamoDB に対するコールのサブセグメント

```
{
  "id": "24756640c0d0978a",
  "start_time": 1.480305974194E9,
  "end_time": 1.4803059742E9,
  "name": "DynamoDB",
  "namespace": "aws",
  "http": {
    "response": {
      "content_length": 60,
      "status": 200
    }
  },
  "aws": {
    "table_name": "scorekeep-user",
    "operation": "UpdateItem",
    "request_id": "UBQNSO5AEM8T4FDA4RQDEB94OVTDRVV4K4HIRGVJF66Q9ASUAAJG",
  }
}
```

名前付きリソースにアクセスしたとき、次のサービスをコールすると、サービスマップに追加のノードが作成されます。特定のリソースをターゲットとしないコールでは、サービスの汎用ノードが作成されます。

- Amazon DynamoDB – テーブル名
- Amazon Simple Storage Service – バケットとキー名
- Amazon Simple Queue Service – キュー名

X-Ray SDK for Python を使用してダウンストリーム HTTP ウェブサービスの呼び出しをトレースする

アプリケーションがマイクロサービスまたはパブリック HTTP API に呼び出しを実行する場合に、X-Ray SDK for Python を使用してこれらの呼び出しを計測し、API をダウンストリームサービスとしてサービスグラフに追加できます。

HTTP クライアントを設定するには、送信呼び出しに使用するライブラリにパッチを適用します。`requests` や Python で実装されている HTTP クライアントを使っている場合は、必要な作業はこれだけです。`aiohttp` の場合は、非同期コンテキストを使用してレコーダーも設定します。

aiohttp 3 のクライアント API を使用する場合は、SDK で提供されるトレース設定のインスタンスを使用して、ClientSession を設定する必要もあります。

Example **aiohttp** 3 クライアント API

```
from aws_xray_sdk.ext.aiohttp.client import aws_xray_trace_config

async def foo():
    trace_config = aws_xray_trace_config()
    async with ClientSession(loop=loop, trace_configs=[trace_config]) as session:
        async with session.get(url) as resp
            await resp.read()
```

ダウンストリームウェブ API に対する呼び出しを計測すると、X-Ray SDK for Python は HTTP リクエストおよびレスポンスに関する情報を含むセグメントを記録します。X-Ray はサブセグメントを使用してリモート API の推測セグメントを生成します。

Example ダウンストリーム HTTP 呼び出しのサブセグメント

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

Example ダウンストリーム HTTP 呼び出しの推定セグメント

```
{
  "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
}
```

X-Ray SDK for Python を使用したカスタムサブセグメントの生成

サブセグメントは、トレースの セグメント をリクストに対応するために行われた作業の詳細で拡張します。計測済みクライアント内で呼び出しを行うたびに、X-Ray SDK によってサブセグメントに生成された情報が記録されます。追加のサブセグメントを作成して、他のサブセグメントをグループ化したり、コードセクションのパフォーマンスを測定したり、注釈とメタデータを記録したりできます。

サブセグメントを管理するには、begin_subsegment および end_subsegment メソッドを使用します。

Example main.py - カスタムサブセグメント

```
from aws_xray_sdk.core import xray_recorder

subsegment = xray_recorder.begin_subsegment('annotations')
subsegment.put_annotation('id', 12345)
xray_recorder.end_subsegment()
```

同期関数のサブセグメントを作成するには、@xray_recorder.capture デコレータを使用します。サブセグメントの名前をキャプチャ関数に渡すことも、関数名を使用することもできます。

Example main.py - 関数サブセグメント

```
from aws_xray_sdk.core import xray_recorder
```

```
@xray_recorder.capture('## create_user')
def create_user():
...
```

非同期関数の場合、@xray_recorder.capture_async デコレータを使用し、非同期コンテキストをレコーダーに渡します。

Example main.py - 非同期関数サブセグメント

```
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()
```

セグメントまたは別のサブセグメント内にサブセグメントを作成する場合、X-Ray SDK for Python によってその ID が生成され、開始時刻と終了時刻が記録されます。

Example サブセグメントとメタデータ

```
"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"
    }
  },
```

X-Ray SDK for Python を使用してセグメントに注釈とメタデータを追加する

アノテーションとメタデータを使用して、リクエスト、環境、またはアプリケーションに関する追加情報を記録できます。X-Ray SDK が作成するセグメントまたは作成するカスタムサブセグメントに、注釈およびメタデータを追加できます。

注釈は文字列、数値、またはブール値を使用したキーと値のペアです。注釈は、フィルタ式用にインデックス付けされます。注釈を使用して、コンソールでトレースをグループ化するため、またはGetTraceSummaries API を呼び出すときに使用するデータを記録します。

メタデータは、オブジェクトとリストを含む、任意のタイプの値を持つことができるキーバリューのペアですが、フィルタ式に使用するためにインデックスは作成されません。メタデータを使用してトレースに保存する追加のデータを記録しますが、検索で使用する必要はありません。

注釈とメタデータに加えて、セグメントにユーザー ID 文字列を記録することもできます。ユーザー ID はセグメントの個別のフィールドに記録され、検索用にインデックスが作成されます。

セクション

- X-Ray SDK for Python で注釈を記録する
- X-Ray SDK for Python でメタデータを記録する
- X-Ray SDK for Python でユーザー ID を記録する

X-Ray SDK for Python で注釈を記録する

注釈を使用して、検索用にインデックスを作成するセグメントまたはサブセグメントに情報を記録します。

注釈の要件

- キー — X-Ray アノテーションのキーには、最大 500 文字の英数字を使用できます。アンダースコア記号 (_) 以外のスペースや記号は使用できません。
- 値 — X-Ray アノテーションの値には、最大 1,000 文字の Unicode 文字を使用できます。
- 注釈の数 — 1 つのトレース 1 つにつき最大 50 個の注釈を使用できます。

注釈を記録するには

1. `xray_recorder` から現在のセグメントまたはサブセグメントの参照を取得します。

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

   または

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

2. 文字列キー、および、ブール値、数値、文字列値を使用して put_annotation を呼び出します。

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

または、put_annotation メソッドを xray_recorder で使用できます。このメソッドは、現在のサブセグメントの注釈を記録します。サブセグメントが開いていない場合は、セグメントの注釈を記録します。

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

SDK は、セグメントドキュメントの annotations オブジェクトにキーと値のペアとして、注釈を記録します。同じキーで put_annotation を 2 回呼び出すと、同じセグメントまたはサブセグメントに以前記録された値が上書きされます。

特定の値を持つ注釈のあるトレースを見つけるには、annotations.*key*フィルタ式の キーワードを使用します。

X-Ray SDK for Python でメタデータを記録する

メタデータを使用して、検索用にインデックスを作成する必要のないセグメントまたはサブセグメントに情報を記録します。メタデータ値は、文字列、数値、ブール値、または JSON オブジェクトや JSON 配列にシリアル化できる任意のオブジェクトになります。

メタデータを記録するには

1. xray_recorder から現在のセグメントまたはサブセグメントの参照を取得します。

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

   または

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

2. 文字列キー、ブール値、数値、文字列値、オブジェクト値、文字列名前空間を使用して put_metadata を呼び出します。

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

または

キーと値だけを使用して put_metadata を呼び出します。

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

または、put_metadata メソッドを xray_recorder で使用できます。このメソッドは、現在のサブセグメントのメタデータを記録します。サブセグメントが開いていない場合は、セグメントのメタデータを記録します。

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

名前空間を指定しない場合、SDK は default を使用します。同じキーで put_metadata を 2 回呼び出すと、同じセグメントまたはサブセグメントに以前記録された値が上書きされます。

X-Ray SDK for Python でユーザー ID を記録する

リクエストセグメントにユーザー ID を記録して、リクエストを送信したユーザーを識別します。

ユーザー ID を記録するには

1. xray_recorder から現在のセグメントへの参照を取得します。

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

2. リクエストを送信したユーザーの文字列 ID を使用して setUser を呼び出します。

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

コントローラーで set_user を呼び出し、アプリケーションがリクエストの処理を開始するとすぐに、ユーザー ID を記録できます。

ユーザー ID のトレースを見つけるには、userフィルタ式で、 キーワードを使用します。

サーバーレス環境にデプロイされたウェブフレームワークの計測

AWS X-Ray SDK for Python は、サーバーレスアプリケーションにデプロイされたウェブフレームワークの計測をサポートしています。サーバーレスはクラウドのネイティブアーキテクチャで、運用上の多くの責任を AWSにシフトさせることができるため、俊敏性とイノベーションを強化できます。

サーバーレスアーキテクチャは、サーバーを意識せずにアプリケーションやサービスを構築および実行できるソフトウェアアプリケーションモデルです。サーバーまたはクラスターのプロビジョニング、パッチ適用、オペレーティングシステムのメンテナンス、キャパシティのプロビジョニングといったインフラストラクチャ管理のタスクが不要になります。サーバーレスアプリケーションは、ほぼすべてのタイプのアプリケーションやバックエンドサービス向けに構築でき、高可用性を実現しながら、アプリケーションの実行やスケーリングに必要な作業のすべてをユーザーに代わって行います。

このチュートリアルでは、サーバーレス環境にデプロイされている Flask や Django などのウェブフレームワーク AWS X-Ray で を自動的に計測する方法を示します。アプリケーションの X-Ray インストルメンテーションを使用すると、Amazon API Gateway から関数を介して行われたすべてのダウンストリーム呼び出しと AWS Lambda 、アプリケーションが行う送信呼び出しを表示できます。

X-Ray SDK for Python では、次の Python アプリケーションフレームワークをサポートしています。

- Flask バージョン 0.8 以降
- Django バージョン 1.0 以降

このチュートリアルでは、Lambda にデプロイされ、API Gateway から呼び出されるサーバーレスアプリケーションのサンプルを作成します。このチュートリアルでは、Zappa を使用して、アプリケーションを Lambda に自動的にデプロイし、API Gateway のエンドポイントを設定します。

前提条件

- Zappa
- Python – バージョン 2.7 または 3.6。
- AWS CLI – AWS CLI が、アプリケーションをデプロイする アカウントと AWS リージョン で設定されていることを確認します。
- Pip
- Virtualenv

ステップ 1: 環境を作成する

このステップでは、virtualenv を使用して仮想環境を作成し、アプリケーションをホスティングします。

1. を使用して AWS CLI、アプリケーションのディレクトリを作成します。その新しいディレクトリに変更します。

```
mkdir serverless_application
cd serverless_application
```

2. 次に、新しいディレクトリ内に仮想環境を作成します。アクティベートするには以下のコマンドを使用します。

```
# Create our virtual environment
virtualenv serverless_env

# Activate it
source serverless_env/bin/activate
```

3. X-Ray、Flask、Zappa およびリクエストライブラリをその環境にインストールします。

```
# Install X-Ray, Flask, Zappa, and Requests into your environment
pip install aws-xray-sdk flask zappa requests
```

4. アプリケーションコードを serverless_application ディレクトリに追加します。この例では、Flasks の Hello World の例を基にします。

   serverless_application ディレクトリに my_app.py という名前のファイルを作成します。テキストエディタで、次のコマンドを追加します。このアプリケーションでは、リクエスト

ライブラリを計測し、Flask アプリケーションのミドルウェアにパッチを適用して、エンドポイント '/' を開きます。

```
# 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
```

ステップ 2: Zappa 環境の作成とデプロイ

このステップでは、Zappa を使用して API Gateway のエンドポイントを自動的に設定し、Lambda にデプロイします。

1. serverless_application ディレクトリ内から Zappa を初期化します。この例では、デフォルト設定を使用しましたが、カスタマイズ設定がある場合は Zappa に設定手順が表示されます。

```
zappa init
```

```
What do you want to call this environment (default 'dev'): dev
...
What do you want to call your bucket? (default 'zappa-*******'): zappa-*******
...
```

```
...
It looks like this is a Flask application.
What's the modular path to your app's function?
This will likely be something like 'your_module.app'.
We discovered: my_app.app
Where is your app's function? (default 'my_app.app'): my_app.app
...
Would you like to deploy this application globally? (default 'n') [y/n/
(p)rimary]: n
```

2. X-Ray を有効にします。zappa_settings.json ファイルを開き、例のように表示されていることを確認します。

```
{
    "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. 設定ファイルのエントリとして "xray_tracing": true を追加します。

```
{
    "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. アプリケーションをデプロイします。これにより、API Gateway エンドポイントが設定され、コードは Lambda に更新されます。

```
zappa deploy
```

```
...
Deploying API Gateway..
Deployment complete!: https://**********.execute-api.us-west-2.amazonaws.com/dev
```

ステップ 3: API Gateway 用 X-Ray トレースを有効にする

このステップでは、API Gateway コンソールを使用して、X-Ray トレースを有効にします。

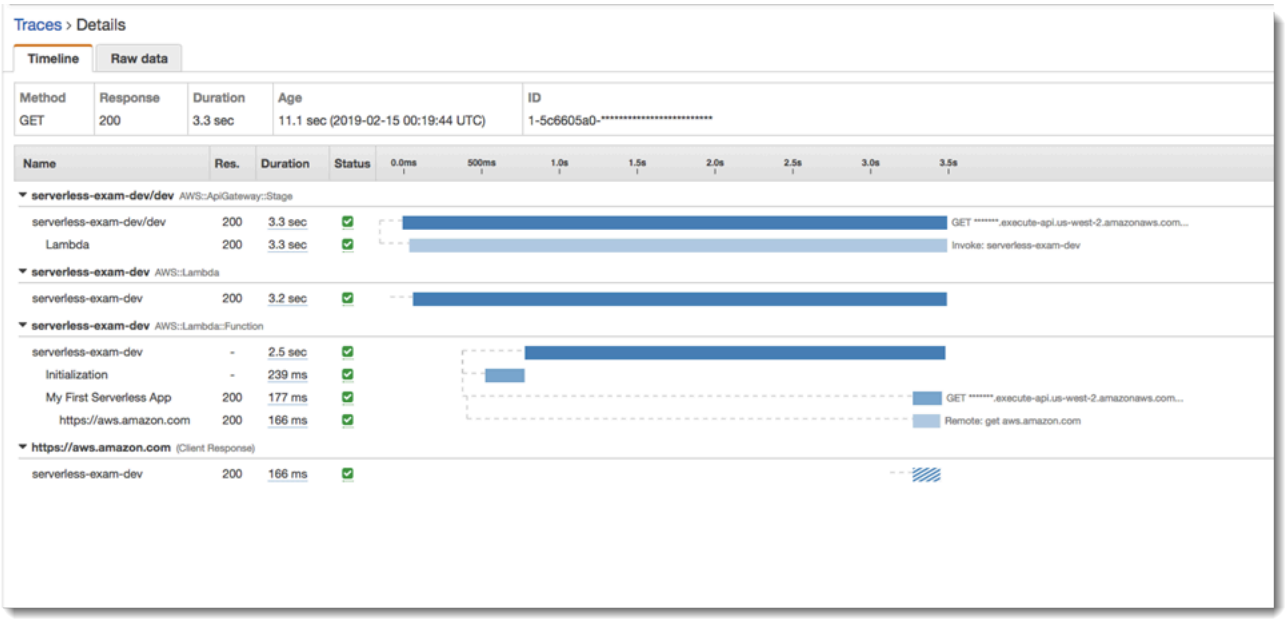
1. にサインイン AWS Management Console し、https://console.aws.amazon.com/apigateway/ で API Gateway コンソールを開きます。
2. 新しく生成された API を探します。`serverless-exam-dev` のようになります。
3. [Stages] (ステージ) を選択します。
4. API のデプロイステージの名前を選択します。デフォルトは `dev` です。
5. [Logs/Tracing] タブで、[X-Ray トレースを有効にする] チェックボックスをオンにします。
6. [変更の保存] をクリックします。
7. ブラウザでエンドポイントにアクセスします。サンプルの `Hello World` アプリケーションを使用した場合は、次のように表示されます。

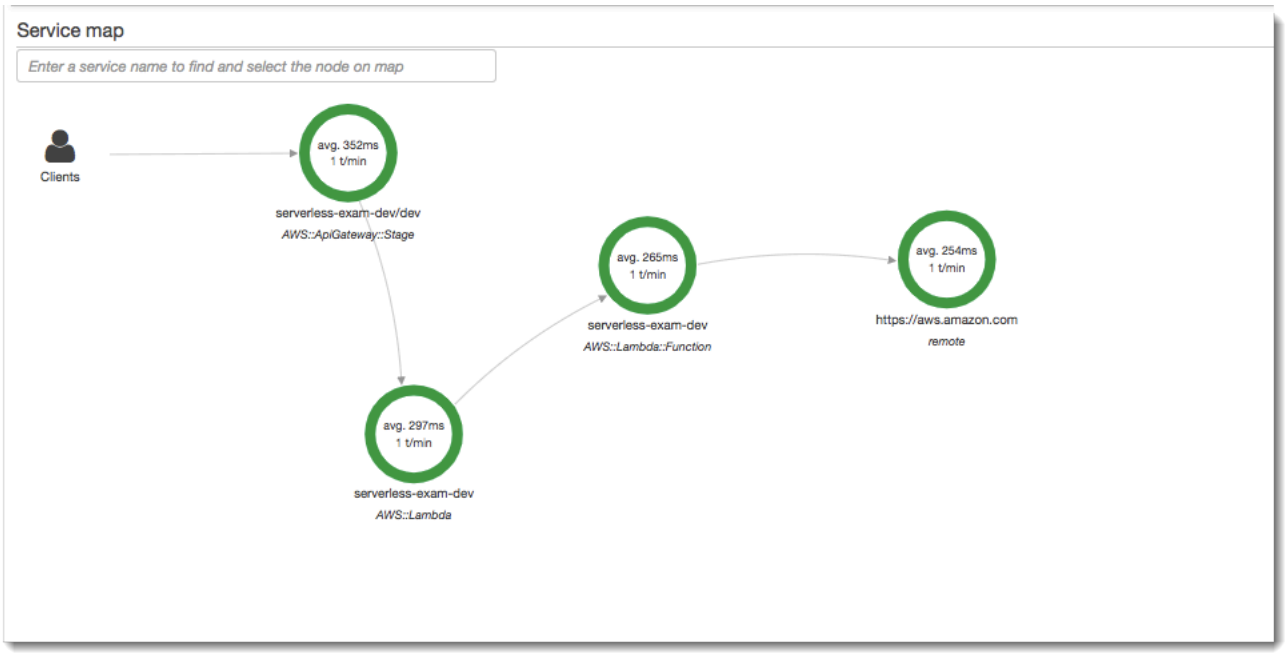
```
"Hello, World: https://aws.amazon.com/"
```

ステップ 4: 作成したトレースを表示する

このステップでは、X-Ray コンソールを使用して、サンプルアプリケーションで作成されたトレースを表示します。トレース解析の詳細なチュートリアルについては、「サービスマップの表示」を参照してください。

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/xray/home で X-Ray コンソールを開きます。
2. API Gateway、Lambda 関数、Lambda コンテナによって生成されたセグメントを表示します。
3. Lambda 関数のセグメントで、`My First Serverless App` という名前のサブセグメントを表示します。次に、`https://aws.amazon.com` という名前の 2 番目のサグメントが続きます。
4. Lambda では、初期化中に `initialization` という名前の 3 番目のサブセグメントが生成されることがあります。





ステップ 5：クリーンアップ

不要なコストがかからないように、使用しなくなったリソースは必ず終了してください。このチュートリアルで示されているように、Zappa のようなツールを使用することで、サーバーレスの再デプロイを効率化することができます。

Lambda、API Gateway、Amazon S3 からアプリケーションを削除するには、 AWS CLIを使用して、プロジェクトディレクトリで次のコマンドを実行します。

```
zappa undeploy dev
```

次のステップ

AWS クライアントを追加し、X-Ray で計測することで、アプリケーションに機能を追加します。サーバーレスコンピューティングのオプションについては、AWSのサーバーレスを参照してください。

を使用してアプリケーションを計測する .NET

X-Ray にトレースを送信するように.NETアプリケーションを計測するには、次の 2 つの方法があります。

- AWS Distro for OpenTelemetry .NET – AWS Distro for OpenTelemetry Collector を介して相関メトリクスとトレースを Amazon CloudWatch、 AWS X-Ray、Amazon OpenSearch Service などの複数の AWS モニタリングソリューションに送信するための一連のオープンソースライブラリを提供するディス AWS トリビューション。
- AWS X-Ray SDK for .NET – X-Ray デーモン を介してトレースを生成して X-Ray に送信するためのライブラリのセット。

詳細については、「AWS Distro for OpenTelemetry と X-Ray SDKs の選択」を参照してください。

AWS のディストリビューション OpenTelemetry .NET

AWS Distro for を使用すると OpenTelemetry .NET、アプリケーションを一度インストゥルメントすれば、相関関係のあるメトリクスとトレースを Amazon CloudWatch、 AWS X-Ray、および Amazon Service AWS などの複数のモニタリングソリューションに送信できます。 OpenSearch X-Ray を AWS Distro for で使用するには、2 OpenTelemetry つのコンポーネントが必要です。1 つは X-Ray で使用できる OpenTelemetry SDK で、もう 1 つは X-Ray で使用可能な OpenTelemetry Collector AWS 用ディストリビューションです。

はじめに、AWS ディストリビューションのドキュメントを参照してください。
OpenTelemetry .NET

OpenTelemetry with AWS X-Ray OpenTelemetry や other 用の AWS Distro の使用について詳しくは、「Distro」を AWS のサービス、AWS ドキュメンテーションについては「Distro」を参照してください。AWS OpenTelemetry

言語のサポートと使用方法について詳しくは、「オブザーバビリティオン」を参照してください AWS 。 GitHub

AWS X-Ray SDK for .NET

X-Ray SDK for .NET は、C# .NET ウェブアプリケーション、.NET Core ウェブアプリケーション、.NET Core 関数を に実装するためのライブラリです AWS Lambda。トレースデータを生成して X-Ray デーモンに送信するためのクラスとメソッドを提供します。これには、アプリケーションによって処理された受信リクエスト、およびアプリケーションがダウンストリーム 、HTTP AWS のサービスウェブ APIs、および SQL データベースに対して行う呼び出しに関する情報が含まれます。

> **Note**
>
> X-Ray SDK for .NET は、オープンソースプロジェクトです。プロジェクトに従い、 で問題とプルリクエストを送信できます GitHub。 github.com/aws/aws-xray-sdk-dotnet

ウェブアプリケーションの場合は、最初にメッセージハンドラーをウェブ設定に追加して、受信リクエストをトレースします。メッセージハンドラーでは、トレース対象リクエストごとに「セグメント」を作成し、レスポンスが送信されるとセグメントを完了します。セグメントが開いている間、SDK クライアントのメソッドを使用してセグメントに情報を追加し、サブセグメントを作成してダウンストリーム呼び出しをトレースできます。また、SDK では、セグメントが開いている間にアプリケーションがスローする例外を自動的に記録します。

インストルメント済みアプリケーションまたはサービスによって呼び出される Lambda 関数の場合、Lambda は トレースヘッダー を読み取り、サンプリングされたリクエストを自動的にトレースします。その他の関数については、Lambda の設定 から受信リクエストのサンプリングとトレースを行うことができます。いずれの場合も、Lambda はセグメントを作成し、X-Ray SDK に提供します。

> **Note**
>
> Lambda では、X-Ray SDK はオプションです。関数でこれを使用しない場合、サービスマップには Lambda サービスのノードと Lambda 関数ごとに 1 つのノードが含まれます。SDK を追加することで、関数コードをインストルメントして、Lambda で記録された関数セグメントにサブセグメントを追加することができます。詳細については、「AWS Lambda および AWS X-Ray」を参照してください。

次に、X-Ray SDK for .NET を使用して次の操作を行います。を計測します AWS SDK for .NET クライアント。計測されたクライアントを使用してダウンストリーム AWS のサービス またはリソースを呼び出すと、SDK はサブセグメントの呼び出しに関する情報を記録します。 AWS サービスおよびサービス内でアクセスするリソースは、トレースマップにダウンストリームノードとして表示され、個々の接続でエラーやスロットリングの問題を特定するのに役立ちます。

X-Ray SDK for .NET は、また、HTTP ウェブ API および SQL データベースに対するダウンストリーム呼び出しの計測もできます。`System.Net.HttpWebRequest` の `GetResponseTraced` 拡張メソッドは送信 HTTP 呼び出しをトレースします。X-Ray SDK for .NET の `SqlCommand` バージョンを使用して SQL クエリを計測します。

SDK のを活用をスタートさせたら、「レコーダーとメッセージハンドラーを設定して」その動作をカスタマイズします。プラグインを追加して、アプリケーションを実行しているコンピューティングリソースに関するデータを記録したり、サンプリングルールを定義することでサンプリングの動作のカスタマイズしたり、アプリケーションログに SDK からの情報をより多くあるいは少なく表示するようにログレベルを設定できます。

アプリケーションが注釈やメタデータで行うリクエストや作業に関する追加情報を記録します。注釈は、フィルタ式で使用するためにインデックス化されたシンプルなキーと値のペアで、特定のデータが含まれているトレースを検索できます。メタデータのエントリは制約が緩やかで、JSON にシリアル化できるオブジェクトと配列全体を記録できます。

> **注釈とメタデータ**
>
> 注釈およびメタデータとは、X-Ray SDK を使用してセグメントに追加する任意のテキストです。注釈は、フィルタ式用にインデックス付けされます。メタデータはインデックス化されませんが、X-Ray コンソールまたは API を使用して raw セグメントで表示できます。X-Ray への読み取りアクセスを許可した人は誰でも、このデータを表示できます。

コードに多数の計測されたクライアントがある場合、単一のリクエストセグメントには計測されたクライアントで行われた呼び出しごとに 1 個の多数のサブセグメントを含めることができます。カスタムサブセグメントで、クライアント呼び出しをラップすることで、サブセグメントを整理してグループできます。関数全体またはコードの任意のセクションのサブセグメントを作成し、親セグメントにすべてのレコードを記述する代わりにサブセグメントにメタデータと注釈を記録できます。

SDK のクラスとメソッドに関するリファレンスドキュメントについては、以下を参照してください。

- AWS X-Ray SDK for .NET API リファレンス
- AWS X-Ray SDK for .NET Core API リファレンス

同じパッケージが .NET および .NET Core の両方をサポートしますが、使用されるクラスは異なります。この章の例は、そのクラスが .NET Core に固有でない限り、.NET API リファレンスにリンクされています。

要件

X-Ray SDK for .NET には、.NET Framework 4.5 以降と が必要です AWS SDK for .NET。

.NET Core アプリケーションと関数の場合、SDK では .NET Core 2.0 以降が必要になります。

.NET X-Ray SDK をアプリケーションに追加する

を使用して NuGet 、X-Ray SDK for .NET をアプリケーションに追加します。

Visual Studio で NuGet パッケージマネージャーを使用して X-Ray SDK for .NET をインストールするには

1. ツール、NuGet パッケージマネージャー、ソリューションの NuGet パッケージの管理を選択します。
2. AWSXRayRecorder を検索します。
3. パッケージを選択し、[Install] を選択します。

依存関係管理

X-Ray SDK for .NETNuGet。SDK の パッケージマネージャーを使ったインストール

```
Install-Package AWSXRayRecorder -Version 2.10.1
```

-AWSXRayRecorder v2.10.1NuGet パッケージには次の依存関係があります。

.NET フレームワーク 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 フレームワーク 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)
```

依存関係管理の詳細については、マイクロソフトのドキュメントを参照してください。NuGet の依存関係そしてNuGet 依存関係の解決。

NET 用 X-Ray SDK の設定

X-Ray SDK for .NET にプラグインを設定して、アプリケーションが実行されているサービスに関する情報が含めたり、デフォルトのサンプリング動作を変更したり、特定のパスに対するリクエストに適用されるサンプリングルールを追加したりできます。

.NET ウェブアプリケーションの場合は、appSettings ファイルの Web.config セクションにキーを追加します。

Example Web.config

```
<configuration>
  <appSettings>
    <add key="AWSXRayPlugins" value="EC2Plugin"/>
    <add key="SamplingRuleManifest" value="sampling-rules.json"/>
  </appSettings>
</configuration>
```

.NET Core の場合は、appsettings.json という最上位のキーを持つ XRay という名前のファイルを作成します。

Example .NET appsettings.json

```
{
  "XRay": {
    "AWSXRayPlugins": "EC2Plugin",
    "SamplingRuleManifest": "sampling-rules.json"
  }
}
```

次に、アプリケーションコードで、設定オブジェクトを構築し、それを使用して X-Ray レコーダーを初期化します。この操作は、レコーダーを初期化する前に実行します。

Example .NET Core Program.cs – Recorder 設定

```
using Amazon.XRay.Recorder.Core;
...
AWSXRayRecorder.InitializeInstance(configuration);
```

.NET Core ウェブアプリケーションを計測する場合は、UseXRayメッセージ ハンドラーを設定するときに、設定オブジェクトを メソッドに渡すこともできます。Lambda 関数の場合は、上記のように InitializeInstance メソッドを使用します。

.NET Core 設定 API の詳細については、docs.microsoft.com の「ASP.NET Core アプリを構成する」を参照してください。

セクション

- プラグイン
- サンプリングルール
- ログ記録 (.NET)
- ログ記録 (.NET Core)
- 環境変数

プラグイン

プラグインを使用して、アプリケーションをホストしているサービスに関するデータを追加します。

プラグイン

- Amazon EC2 – は、インスタンス ID、アベイラビリティーゾーン、および CloudWatch Logs グループEC2Pluginを追加します。
- ElasticBeanstalk– ElasticBeanstalkPluginは、環境名、バージョンラベル、およびデプロイ ID を追加します。
- Amazon ECS —ECSPluginは、コンテナ ID を追加します。

プラグインを使用するには、AWSXRayPlugins 設定を追加して X-Ray SDK for .NET クライアントを設定します。複数のプラグインがアプリケーションに適用される場合は、そのすべてをカンマで区切って同じ設定で指定します。

Example Web.config - プラグイン

```
<configuration>
  <appSettings>
    <add key="AWSXRayPlugins" value="EC2Plugin,ElasticBeanstalkPlugin"/>
  </appSettings>
</configuration>
```

Example .NET Core appsettings.json – プラグイン

```
{
  "XRay": {
    "AWSXRayPlugins": "EC2Plugin,ElasticBeanstalkPlugin"
  }
}
```

サンプリングルール

SDK は X-Ray コンソールで定義したサンプリングルールを使用し、記録するリクエストを決定します。デフォルトルールでは、最初のリクエストを毎秒トレースし、X-Ray にトレースを送信するすべてのサービスで追加のリクエストの 5% をトレースします。X-Ray コンソールに追加のルールを作成するをクリックして、各アプリケーションで記録されるデータ量をカスタマイズします。

SDK は、定義された順序でカスタムルールを適用します。リクエストが複数のカスタムルールと一致する場合、SDK は最初のルールのみを適用します。

> **Note**
>
> SDK が X-Ray に到達してサンプリングルールを取得できない場合、1 秒ごとに受信された最初のリクエストのデフォルトのローカルルールに戻り、ホストあたりの追加リクエストの 5% に戻ります。これは、ホストがサンプリング API を呼び出す権限を持っていない場合や、SDK によって行われる API 呼び出しの TCP プロキシとして機能する X-Ray デーモンに接続できない場合に発生します。

JSON ドキュメントからサンプリングルールをロードするように SDK を設定することもできます。SDK は、X-Ray サンプリングが利用できない場合のバックアップとしてローカルルールを使用することも、ローカルルールを排他的に使用することもできます。

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
  }
}
```

この例では、1 つのカスタムルールとデフォルトルールを定義します。カスタムルールでは、5 パーセントのサンプリングレートが適用され、/api/move/以下のパスに対してトレースするリクエストの最小数はありません。デフォルトのルールでは、1秒ごとの最初のリクエストおよび追加リクエストの 10 パーセントをトレースします。

ルールをローカルで定義することの欠点は、固定ターゲットが X-Ray サービスによって管理されるのではなく、レコーダーの各インスタンスによって個別に適用されることです。より多くのホストをデプロイすると、固定レートが乗算され、記録されるデータ量の制御が難しくなります。

では AWS Lambda、サンプリングレートを変更することはできません。関数がインストルメント化されたサービスによって呼び出された場合、そのサービスによってサンプリングされたリクエストを生成した呼び出しは Lambda によって記録されます。アクティブなトレースが有効で、トレースヘッダーが存在しない場合、Lambda はサンプリングを決定します。

バックアップルールを設定するには、SamplingRuleManifest 設定を使用して X-Ray SDK for .NET にファイルからサンプリングルールをロードするように指示します。

Example .NET Web.config - サンプリングルール

```
<configuration>
  <appSettings>
```

```
    <add key="SamplingRuleManifest" value="sampling-rules.json"/>
  </appSettings>
</configuration>
```

Example .NET Core appsettings.json – サンプリングルール

```
{
  "XRay": {
    "SamplingRuleManifest": "sampling-rules.json"
  }
}
```

ローカルルールのみを使用するには、LocalizedSamplingStrategy でレコーダーをビルドします。バックアップルールが設定されている場合、その設定を削除します。

Example .NET global.asax – ローカルサンプリングルール

```
var recorder = new AWSXRayRecorderBuilder().WithSamplingStrategy(new
 LocalizedSamplingStrategy("samplingrules.json")).Build();
AWSXRayRecorder.InitializeInstance(recorder: recorder);
```

Example .NET Core Program.cs – Local サンプリングルール

```
var recorder = new AWSXRayRecorderBuilder().WithSamplingStrategy(new
 LocalizedSamplingStrategy("sampling-rules.json")).Build();
AWSXRayRecorder.InitializeInstance(configuration,recorder);
```

ログ記録 (.NET)

X-Ray SDK for .NET では、AWS SDK for .NET。 AWS SDK for .NET 出力をログに記録するようにアプリケーションを設定済みである場合は、X-Ray SDK for .NET からの出力にも同じ設定が適用されます。

ログ記録を設定するには、aws ファイルまたは App.config ファイルに、Web.config という名前の設定セクションを追加します。

Example Web.config - ログ記録

```
...
<configuration>
  <configSections>
```

```
    <section name="aws" type="Amazon.AWSSection, AWSSDK.Core"/>
  </configSections>
  <aws>
    <logging logTo="Log4Net"/>
  </aws>
</configuration>
```

詳細については、『AWS SDK for .NET 開発者ガイド』の「AWS SDK for .NET アプリケーションの設定」を参照してください。

ログ記録 (.NET Core)

X-Ray SDK for .NET では、AWS SDK for .NET。.NET Core アプリケーションのロギングを構成するには、logging オプションを`AWSXRayRecorder.RegisterLogger`方法。

たとえば、log4net を使用するには、ロガー、出力形式、およびファイルの場所を定義する設定ファイルを作成します。

Example .NET Core log4net.config

```
<?xml version="1.0" encoding="utf-8" ?>
<log4net>
  <appender name="FileAppender" type="log4net.Appender.FileAppender,log4net">
    <file value="c:\logs\sdk-log.txt" />
    <layout type="log4net.Layout.PatternLayout">
      <conversionPattern value="%date [%thread] %level %logger - %message%newline" />
    </layout>
  </appender>
  <logger name="Amazon">
    <level value="DEBUG" />
    <appender-ref ref="FileAppender" />
  </logger>
</log4net>
```

次に、ロガーを作成し、プログラムコードで設定を適用します。

Example .NET Core Program.cs – ロギング

```
using log4net;
using Amazon.XRay.Recorder.Core;

class Program
```

```
{
  private static ILog log;
  static Program()
  {
    var logRepository = LogManager.GetRepository(Assembly.GetEntryAssembly());
    XmlConfigurator.Configure(logRepository, new FileInfo("log4net.config"));
    log = LogManager.GetLogger(typeof(Program));
    AWSXRayRecorder.RegisterLogger(LoggingOptions.Log4Net);
  }
  static void Main(string[] args)
  {
  ...
  }
}
```

log4net の設定の詳細については、logging.apache.org で設定を参照してください。

環境変数

環境変数を使用して、X-Ray SDK を .NET 用に設定できます。SDK は次の変数をサポートしています。

- AWS_XRAY_TRACING_NAME – SDK がセグメントに使用するサービス名を設定します。サーブレットフィルタのセグメント命名ルールで設定したサービス名を上書きします。
- AWS_XRAY_DAEMON_ADDRESS – X-Ray デーモン リスナーのホストとポートを設定します。デフォルトでは、SDK はトレースデータ (UDP) とサンプリング (TCP) の両方に127.0.0.1:2000を使用します。この変数は、デーモンを次のように構成している場合に使用します。別のポートでリッスンするまたは、別のホストで実行されている場合。

  [形式]

  - 同じポート – *address*:*port*
  - 異なるポート – tcp:*address*:*port* udp:*address*:*port*
- AWS_XRAY_CONTEXT_MISSING – 計測されたコードが、セグメントが開いていないときにデータを記録しようとした場合に例外をスローするには、RUNTIME_ERROR に設定します。

  有効な値

  - RUNTIME_ERROR— ランタイム例外をスローします。
  - LOG_ERROR – エラーをログ記録して続行します (デフォルト)。
  - IGNORE_ERROR – エラーを無視して続行します。

オープン状態のリクエストがない場合、または新しいスレッドを発生させるコードで、スタートアップコードに実装されたクライアントを使用しようとした場合に発生する可能性がある、セグメントまたはサブセグメントの欠落に関連するエラー。

X-Ray SDK for .NET

X-Ray SDK を使用して、アプリケーションが Amazon EC2、AWS Elastic Beanstalk、または Amazon ECS の EC2 インスタンスで処理する受信 HTTP リクエストをトレースできます。

メッセージハンドラーを使用して受信 HTTP リクエストを計測します。X-Ray メッセージハンドラーをアプリケーションに追加すると、サンプリングされた各リクエストに X-Ray SDK for .NET によってセグメントが作成されます。このセグメントには、時間、メソッド、HTTP リクエストの処理などが含まれます。追加の計測により、このセグメントでサブセグメントが作成されます。

> **Note**
>
> AWS Lambda関数では、Lambda は、サンプリングされた各リクエストのセグメントを作成します。詳細については、「AWS Lambda および AWS X-Ray」を参照してください。

各セグメントには、サービスマップ内のアプリケーションを識別する名前があります。セグメントの名前は静的に指定することも、受信リクエストのホストヘッダーに基づいて動的に名前を付けるように SDK を設定することもできます。動的ネーミングでは、リクエスト内のドメイン名に基づいてトレースをグループ化でき、名前が予想されるパターンと一致しない場合 (たとえば、ホストヘッダーが偽造されている場合) 、デフォルト名を適用できます。

> **転送されたリクエスト**
>
> ロードバランサーまたは他の仲介者がアプリケーションにリクエストを転送する場合、X-Ray は、クライアントの IP をIP パケットの送信元 IP からではなく、リクエストのX-Forwarded-Forヘッダーから取得します。転送された要求に対して記録されたクライアント IP は偽造される可能性があるため、信頼されるべきではありません。

メッセージハンドラーは、次の情報が含まれる `http` ブロックを使用して、各受信リクエスト用にセグメントを作成します。

- HTTP メソッド – GET、POST、PUT、DELETE、その他。

- クライアントアドレス – リクエストを送信するクライアントの IP アドレス。
- レスポンスコード – 完了したリクエストの HTTP レスポンスコード。
- タイミング – 開始時間 (リクエストが受信された時間) および終了時間 (レスポンスが送信された時間)。
- ユーザーエージェント — リクエストからのuser-agent
- コンテンツの長さ — レスポンスからのcontent-length

セクション

受信リクエストの計測 (.NET)

アプリケーションによって処理されるリクエストを計測するには、RegisterXRay ファイルの Init メソッドで global.asax を呼び出します。

Example global.asax - メッセージハンドラー

```
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");
    }
  }
}
```

受信リクエストの計測 (.NET Core)

アプリケーションによって処理されるリクエストを計測するには、UseXRay他のミドルウェアより前のメソッドConfigureStartup クラスのメソッドは、理想的にはX-Rayミドルウェアがリクエスト

を処理する最初のミドルウェアであり、パイプラインでレスポンスを処理する最後のミドルウェアにする必要があります。

> **Note**
>
> .NET Core 2.0 の場合、UseExceptionHandlerアプリケーションのメソッドで、必ず呼び出してくださいUseXRay後UseExceptionHandlerメソッドを使用して、例外が記録されるようにします。

Example Startup.cs

<caption>.NET Core 2.1 and above</caption>

```
using Microsoft.AspNetCore.Builder;

public void Configure(IApplicationBuilder app, IHostingEnvironment env)
  {
    app.UseXRay("MyApp");
    // additional middleware
    ...
  }
```

<caption>.NET Core 2.0</caption>

```
using Microsoft.AspNetCore.Builder;

public void Configure(IApplicationBuilder app, IHostingEnvironment env)
  {
    app.UseExceptionHandler("/Error");
    app.UseXRay("MyApp");
    // additional middleware
    ...
  }
```

この UseXRay メソッドは、2 番目の引数として設定オブジェクトを受け取ることもできます。

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

セグメント命名ルールの設定

AWS X-Rayはサービス名を使用してアプリケーションを識別し、他のアプリケーション、データベース、外部 API、およびAWSアプリケーションが使用するリソースと区別します。X-Ray SDK が受信リクエストのセグメントを生成すると、アプリケーションのサービス名がセグメントの名前フィールドに記録されます。

X-Ray SDK では、HTTP リクエストヘッダーのホスト名の後にセグメントの名前を指定できます。ただし、このヘッダーは偽造され、サービスマップに予期しないノードが発生する可能性があります。偽造されたホストヘッダーを持つリクエストによって SDK がセグメントの名前を間違えないようにするには、受信リクエストのデフォルト名を指定する必要があります。

アプリケーションが複数のドメインのリクエストを処理する場合、動的ネーミングストラテジーを使用してセグメント名にこれを反映するように SDK を設定できます。動的ネーミングストラテジーにより、SDK は予想されるパターンに一致するリクエストにホスト名を使用し、そうでないリクエストにデフォルト名を適用できます。

たとえば、3 つのサブドメイン (`www.example.com`,`api.example.com`,および`static.example.com`) に対してリクエストを処理する単一のアプリケーションがあるとします。動的ネーミングストラテジーをパターン `*.example.com` で使用して、異なる名前を持つ各サブドメインのセグメントを識別することができます。結果的にはサービスマップ上に 3 つのサービスノードを作成することになります。アプリケーションがパターンと一致しないホスト名のリクエストを受信すると、指定したフォールバック名を持つ 4 番目のノードがサービスマップに表示されます。

すべてのリクエストセグメントに対して同じ名前を使用するには、前のセクションで示すとおり、メッセージハンドラを初期化するときに、アプリケーションの名前を指定します。これは、`FixedSegmentNamingStrategy` を作成して、`RegisterXRay` メソッドに渡すのと同じ効果があります。

```
AWSXRayASPNET.RegisterXRay(this, new FixedSegmentNamingStrategy("MyApp"));
```

> **Note**
>
> コードで定義したデフォルトのサービス名は、`AWS_XRAY_TRACING_NAME` 環境変数で上書きできます。

動的な命名戦略は、ホスト名と一致するようパターンを定義し、HTTP リクエストのホスト名がパターンと一致しない場合はデフォルトの名前を使用します。動的にセグメントに命名するには、DynamicSegmentNamingStrategy を作成して、RegisterXRay メソッドに渡します。

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

X-Ray AWS SDK for .NET を使用した SDK 呼び出しのトレース

アプリケーションが を呼び出し AWS のサービス てデータの保存、キューへの書き込み、または通知の送信を行う場合、X-Ray SDK for .NET はサブセグメント の呼び出しダウンストリームを追跡します。これらのサービス (Amazon S3 バケットや Amazon SQS キューなど) 内でアクセスするトレースされた AWS のサービス および リソースは、X-Ray コンソールのトレースマップにダウンストリームノードとして表示されます。

クライアントを作成するRegisterXRayForAllServices前に を呼び出すことで、すべての AWS SDK for .NET クライアントを計測できます。

Example SampleController.cs - DynamoDB クライアント計測

```
using Amazon;
using Amazon.Util;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DocumentModel;
using Amazon.XRay.Recorder.Core;
using Amazon.XRay.Recorder.Handlers.AwsSdk;

namespace SampleEBWebApplication.Controllers
{
  public class SampleController : ApiController
  {
    AWSSDKHandler.RegisterXRayForAllServices();
    private static readonly Lazy<AmazonDynamoDBClient> LazyDdbClient = new
 Lazy<AmazonDynamoDBClient>(() =>
    {
      var client = new AmazonDynamoDBClient(EC2InstanceMetadata.Region ??
 RegionEndpoint.USEast1);
      return client;
    });
```

一部のサービスのクライアントのみを計測するには、RegisterXRayForAllServices ではなく RegisterXRay を呼び出します。強調表示されたテキストを、サービスのクライアントインターフェイスの名前で置き換えます。

```
AWSSDKHandler.RegisterXRay<IAmazonDynamoDB>()
```

すべてのサービスで、X-Ray コンソールで呼び出される API の名前を確認できます。サービスのサブセットの場合、X-Ray SDK はセグメントに情報を追加して、サービスマップでより細かく指定します。

たとえば、実装された DynamoDB クライアントでコールすると、SDK はテーブルをターゲットとするコールのセグメントにテーブル名を追加します。コンソールで、各テーブルはサービスマップ内に個別のノードとして表示され、テーブルをターゲットにしないコール用の汎用の DynamoDB ノードが表示されます。

Example 項目を保存するための DynamoDB に対するコールのサブセグメント

```
{
  "id": "24756640c0d0978a",
  "start_time": 1.480305974194E9,
  "end_time": 1.4803059742E9,
  "name": "DynamoDB",
  "namespace": "aws",
  "http": {
    "response": {
      "content_length": 60,
      "status": 200
    }
  },
  "aws": {
    "table_name": "scorekeep-user",
    "operation": "UpdateItem",
    "request_id": "UBQNSO5AEM8T4FDA4RQDEB94OVTDRVV4K4HIRGVJF66Q9ASUAAJG",
  }
}
```

名前付きリソースにアクセスしたとき、次のサービスをコールすると、サービスマップに追加のノードが作成されます。特定のリソースをターゲットとしないコールでは、サービスの汎用ノードが作成されます。

- Amazon DynamoDB – テーブル名

- Amazon Simple Storage Service – バケットとキー名
- Amazon Simple Queue Service – キュー名

X-Ray SDK for .NET を使用してダウンストリーム HTTP ウェブサービスの呼び出しをトレースする

アプリケーションがマイクロサービスまたはパブリック HTTP API に呼び出しを実行する場合に、`GetResponseTraced` で X-Ray SDK for .NET の `System.Net.HttpWebRequest` 拡張メソッドを使用してこれらの呼び出しを計測し、API をダウンストリームサービスとしてサービスグラフに追加できます。

Example HttpWebRequest

```
using System.Net;
using Amazon.XRay.Recorder.Core;
using Amazon.XRay.Recorder.Handlers.System.Net;

private void MakeHttpRequest()
{
  HttpWebRequest request = (HttpWebRequest)WebRequest.Create("http://names.example.com/
api");
  request.GetResponseTraced();
}
```

非同期呼び出しには、`GetAsyncResponseTraced` を使用します。

```
request.GetAsyncResponseTraced();
```

`system.net.http.httpclient` を使用する場合は、`HttpClientXRayTracingHandler` 委任ハンドラを使用して呼び出しを記録します。

Example HttpClient

```
using System.Net.Http;
using Amazon.XRay.Recorder.Core;
using Amazon.XRay.Recorder.Handlers.System.Net;

private void MakeHttpRequest()
{
```

```
  var httpClient = new HttpClient(new HttpClientXRayTracingHandler(new
 HttpClientHandler()));
  httpClient.GetAsync(URL);
}
```

ダウンストリームウェブ API に対する呼び出しを計測すると、X-Ray SDK for .NET は HTTP リクエストおよびレスポンスに関する情報を含むセグメントを記録します。X-Ray はサブセグメントを使用して API の推測セグメントを生成します。

Example ダウンストリーム HTTP 呼び出しのサブセグメント

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

Example ダウンストリーム HTTP 呼び出しの推定セグメント

```
{
  "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
}
```

X-Ray SDK for .NET

X-Ray SDK for .NET は、SqlCommand の代わりに使用できる TraceableSqlCommand という名前の System.Data.SqlClient.SqlCommand のラッパークラスを提供します。TraceableSqlCommand クラスを使用して、SQL コマンドを初期化できます。

同期メソッドと非同期メソッドを使用した SQL クエリのトレース

以下の例では、TraceableSqlCommand を使用して自動的に SQL Server クエリを同期的および非同期的にトレースする方法を示しています。

Example **Controller.cs** - SQL クライアント計測 (同期)

```
using Amazon;
using Amazon.Util;
using Amazon.XRay.Recorder.Core;
using Amazon.XRay.Recorder.Handlers.SqlServer;

private void QuerySql(int id)
{
  var connectionString = ConfigurationManager.AppSettings["RDS_CONNECTION_STRING"];
  using (var sqlConnection = new SqlConnection(connectionString))
  using (var sqlCommand = new TraceableSqlCommand("SELECT " + id, sqlConnection))
  {
    sqlCommand.Connection.Open();
    sqlCommand.ExecuteNonQuery();
  }
}
```

ExecuteReaderAsync メソッドを使用して、クエリを非同期的に実行できます。

Example **Controller.cs** - SQL クライアント計測 (非同期)

```
using Amazon;
```

```
using Amazon.Util;
using Amazon.XRay.Recorder.Core;
using Amazon.XRay.Recorder.Handlers.SqlServer;
private void QuerySql(int id)
{
  var connectionString = ConfigurationManager.AppSettings["RDS_CONNECTION_STRING"];
  using (var sqlConnection = new SqlConnection(connectionString))
  using (var sqlCommand = new TraceableSqlCommand("SELECT " + id, sqlConnection))
  {
    await sqlCommand.ExecuteReaderAsync();
  }
}
```

SQL Server に対して実行された SQL クエリの収集

SQL クエリによって作成されたサブセグメントの一部として SqlCommand.CommandText のキャプチャを有効にできます。SqlCommand.CommandText は、サブセグメント JSON のフィールド sanitized_query として表示されます。デフォルトでは、この機能はセキュリティのために無効になっています。

> **Note**
>
> SQL クエリに機密情報をクリアテキストとして含める場合は、収集機能を有効にしないでください。

SQL クエリの収集を有効にするには、以下の 2 つの方法があります。

- アプリケーションのグローバル設定で CollectSqlQueries プロパティを true に設定する。
- インスタンス内の呼び出しを収集するには、TraceableSqlCommand インスタンスの collectSqlQueries パラメータを true に設定する。

Global CollectSqlQueries プロパティを有効にする

以下の例では、.NET および .NET Core の CollectSqlQueries プロパティを有効にする方法を示しています。

.NET

NET のアプリケーションのグローバル設定で CollectSqlQueries プロパティを true に指定するには、ここで示しているように App.config または Web.config ファイルの appsettings を変更します。

Example **App.config** または **Web.config** – SQL クエリの収集をグローバルに有効にする

```
<configuration>
<appSettings>
    <add key="CollectSqlQueries" value="true">
</appSettings>
</configuration>
```

.NET Core

.NET Core のアプリケーションのグローバル設定で CollectSqlQueries プロパティを true に指定するには、ここで示しているように appsettings.jsonX-Ray キーの ファイルを変更します。

Example **appsettings.json** – SQL クエリの収集をグローバルに有効にする

```
{
  "XRay": {
    "CollectSqlQueries":"true"
  }
}
```

collectSqlQueries パラメータを有効にする

TraceableSqlCommand インスタンスの collectSqlQueries パラメータを true に設定することで、そのインスタンスを使用して実行された SQL Server クエリの SQL クエリテキストを収集できます。このパラメータを false に設定すると、TraceableSqlCommand インスタンスの CollectSqlQuery 機能が無効になります。

> **Note**
>
> TraceableSqlCommand インスタンスの collectSqlQueries の値は、CollectSqlQueries プロパティのグローバル設定で指定された値を上書きします。

Example サンプル **Controller.cs** – インスタンスの SQL クエリの収集を有効にする

```
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();
  }
}
```

追加のサブセグメントを作成する

サブセグメントはトレースを拡張しますセグメントリクエストを処理するために行われた作業の詳細を含む。計測済みクライアント内で呼び出しを行うたびに、X-Ray SDK によってサブセグメントに生成された情報が記録されます。追加のサブセグメントを作成して、他のサブセグメントをグループ化したり、コードセクションのパフォーマンスを測定したり、注釈とメタデータを記録したりできます。

サブセグメントを管理するには、BeginSubsegment および EndSubsegment メソッドを使用します。try ブロックでサブセグメントの任意の作業を実行し、AddException を使用して例外をトレースします。ブロックで EndSubsegment を呼び出し、サブセグメントが閉じられたことを確認します。finally

Example Controller.cs – カスタムサブセグメント

```
AWSXRayRecorder.Instance.BeginSubsegment("custom method");
try
{
  DoWork();
}
catch (Exception e)
{
  AWSXRayRecorder.Instance.AddException(e);
```

```
}
finally
{
  AWSXRayRecorder.Instance.EndSubsegment();
}
```

セグメントまたは別のサブセグメント内にサブセグメントを作成する場合、X-Ray SDK for .NET によってその ID が生成され、開始時刻と終了時刻が記録されます。

Example サブセグメントとメタデータ

```
"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"
    }
  },
```

X-Ray SDK for .NET を使用してセグメントに注釈とメタデータを追加する

注釈とメタデータを使用して、リクエスト、環境、またはアプリケーションに関する追加情報を記録できます。X-Ray SDK が作成するセグメントまたは作成するカスタムサブセグメントに、注釈およびメタデータを追加できます。

注釈は文字列、数値、またはブール値を使用したキーと値のペアです。注釈は、フィルタ式用にインデックス付けされます。注釈を使用して、コンソールでトレースをグループ化するため、または`GetTraceSummaries` API を呼び出すときに使用するデータを記録します。

メタデータは、オブジェクトとリストを含む、任意のタイプの値を持つことができるキーバリューのペアですが、フィルタ式に使用するためにインデックスは作成されません。メタデータを使用してトレースに保存する追加のデータを記録しますが、トレースの検索用に使用する必要はありません。

セクション

- X-Ray SDK を使用して.NET の注釈を記録する
- X-Ray SDK for .NET

X-Ray SDK を使用して.NET の注釈を記録する

注釈を使用して、検索用にインデックスを作成するセグメントまたはサブセグメントに情報を記録します。

X-Ray のすべての注釈には、以下が必要です。

注釈の要件

- キー – X-Ray 注釈のキーには、最大 500 文字の英数字を使用できます。アンダースコア記号 (_) 以外のスペースや記号は使用できません。
- 値 – X-Ray 注釈の値は、最大 1,000 文字の Unicode 文字を持つことができます。
- 注釈の数 — トレースごとに最大 50 個の注釈を使用できます。

AWS Lambda 関数の外部に注釈を記録するには

1. `AWSXRayRecorder` のインスタンスを取得します。

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

2. 文字列キー、およびブール値、Int32、Int64、Double、文字列値を使用して `addAnnotation` を呼び出します。

```
recorder.AddAnnotation("mykey", "my value");
```

AWS Lambda 関数内に注釈を記録するには

Lambda 関数内のセグメントとサブセグメントの両方が Lambda ランタイム環境によって管理されます。Lambda 関数内のセグメントまたはサブセグメントに注釈を追加する場合は、次の操作を行う必要があります。

1. Lambda 関数内にセグメントまたはサブセグメントを作成します。
2. セグメントまたはサブセグメントに注釈を追加します。
3. セグメントまたはサブセグメントを終了します。

次のコード例は、Lambda 関数内のサブセグメントに注釈を追加する方法を示しています。

```
#Create the subsegment
AWSXRayRecorder.Instance.BeginSubsegment("custom method");
#Add an annotation
AWSXRayRecorder.Instance.AddAnnotation("My", "Annotation");
try
{
  YourProcess(); #Your function
}
catch (Exception e)
{
  AWSXRayRecorder.Instance.AddException(e);
}
finally #End the subsegment
{
  AWSXRayRecorder.Instance.EndSubsegment();
}
```

X-Ray SDK は、セグメントドキュメント内のannotationsオブジェクトにキーと値のペアとして注釈を記録します。同じキーで addAnnotationオペレーションを 2 回呼び出すと、同じセグメントまたはサブセグメントで以前に記録された値が上書きされます。

特定の値を持つ注釈を持つトレースを検索するには、フィルター式で annotations.*key*キーワードを使用します。詳細については、「フィルター式を使用する」を参照してください。

X-Ray SDK for .NET

メタデータを使用して、検索内で使用するためにインデックスを作成する必要がないセグメントまたはサブセグメントの情報を記録します。メタデータ値は、文字列、数値、ブール値、または JSON オブジェクトまたは配列にシリアル化できるその他のオブジェクトです。

メタデータを記録するには

1. 次のコード例に示すようにAWSXRayRecorder、 のインスタンスを取得します。

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

2. 次のコード例に示すように、AddMetadata文字列名前空間、文字列キー、およびオブジェクト値を使用して を呼び出します。

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

次のコード例に示すように、キーと値のペアのみを使用して AddMetadataオペレーションを呼び出すこともできます。

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

名前空間の値を指定しない場合、X-Ray SDK は を使用しますdefault。同じキーで AddMetadataオペレーションを 2 回呼び出すと、同じセグメントまたはサブセグメントで以前に記録された値が上書きされます。

Ruby を使用してアプリケーションを計測する

X-Ray にトレースを送信する Ruby アプリケーションを計測するには、次の 2 つの方法があります。

- AWS Distro for OpenTelemetry Ruby – AWS Distro for OpenTelemetry Collector を介して相関メトリクスとトレースを Amazon CloudWatch、 AWS X-Ray、Amazon OpenSearch Service などの複数の AWS モニタリングソリューションに送信するための一連のオープンソースライブラリを提供するディス AWS トリビューション。
- AWS X-Ray SDK for Ruby – X-Ray デーモン を介してトレースを生成して X-Ray に送信するためのライブラリのセット。

詳細については、「AWS Distro for OpenTelemetry と X-Ray SDKs の選択」を参照してください。

AWS Distro for OpenTelemetry Ruby

AWS Distro for OpenTelemetry (ADOT) Ruby を使用すると、アプリケーションを一度計測し、相関のあるメトリクスとトレースを Amazon CloudWatch、AWS X-Ray、Amazon OpenSearch Service などの複数の AWS モニタリングソリューションに送信することができます。X-Ray を ADOT で使用するには、X-Ray で使用できる OpenTelemetry SDK と、X-Ray で使用できる AWS Distro for OpenTelemetry Collector の 2 つのコンポーネントが必要です。

開始するには、「AWS Distro for OpenTelemetry Ruby documentation」を参照してください。

AWS Distro for OpenTelemetry と AWS X-Ray やその他の AWS のサービス の併用については、「AWS Distro for OpenTelemetry」または「AWS Distro for OpenTelemetry Documentation」を参照してください。

言語サポートと使用方法の詳細については、「AWS Observability on Github」を参照してください。

AWS X-Ray SDK for Ruby

X-Ray SDKは、Ruby ウェブアプリケーション用のライブラリです。トレースデータを作成して X-Ray デーモンに送信するためのクラスとメソッドを提供します。トレースデータには、アプリケーションによって処理された受信 HTTP リクエスト、および AWS SDK、HTTP クライアント、またはアクティブなレコードクライアントを使用してアプリケーションがダウンストリームサービスに対して行う呼び出しに関する情報が含まれます。セグメントを手動で作成し、注釈およびメタデータにデバッグ情報を追加することもできます。

gemfile に追加し、`bundle install` を実行することで、SDK をダウンロードできます。

Example Gemfile

```
gem 'aws-sdk'
```

Rails を使用する場合は、最初に X-Ray SDK ミドルウェアを追加して受信リクエストをトレースします。リクエストフィルタにより、セグメントが作成されます。セグメントが開いている間、SDK クライアントのメソッドを使用してセグメントに情報を追加し、サブセグメントを作成してダウンストリーム呼び出しをトレースできます。また、SDK では、セグメントが開いている間にアプリケーションがスローする例外を自動的に記録します。Rails 以外のアプリケーションの場合、手動でセグメントを作成することができます。

次に、X-Ray SDK を使用して AWS SDK for Ruby、関連するライブラリにパッチを適用するようにレコーダーを設定することで、、、、HTTP、SQL クライアントを計測します。計測されたクライアントを使用してダウンストリーム AWS のサービス またはリソースを呼び出すと、SDK はサブセグメントの呼び出しに関する情報を記録します。 AWS のサービス サービス内でアクセスするリソースは、トレースマップにダウンストリームノードとして表示され、個々の接続でエラーやスロットリングの問題を識別しやすくなります。

SDK を入手したら、レコーダーを設定して動作をカスタマイズします。プラグインを追加して、アプリケーションを実行しているコンピューティングリソースに関するデータを記録したり、サンプリングルールを定義することでサンプリングの動作のカスタマイズしたり、ロガーを提供してアプリケーションログに SDK からの情報をより多くあるいは少なく表示することができます。

アプリケーションが注釈やメタデータで行うリクエストや作業に関する追加情報を記録します。注釈は、フィルタ式で使用するためにインデックス化されたシンプルなキーと値のペアで、特定のデータが含まれているトレースを検索できます。メタデータのエントリは制約が緩やかで、JSON にシリアル化できるオブジェクトと配列全体を記録できます。

> **注釈とメタデータ**
>
> 注釈およびメタデータとは、X-Ray SDK を使用してセグメントに追加する任意のテキストです。注釈は、フィルタ式用にインデックス付けされます。メタデータはインデックス化されませんが、X-Ray コンソールまたは API を使用して raw セグメントで表示できます。X-Ray への読み取りアクセスを許可した人は誰でも、このデータを表示できます。

コードに多数の計測されたクライアントがある場合、単一のリクエストセグメントには計測されたクライアントで行われた呼び出しごとに 1 個の多数のサブセグメントを含めることができます。カスタムサブセグメントで、クライアント呼び出しをラップすることで、サブセグメントを整理してグループできます。関数全体またはコードの任意のセクションのサブセグメントを作成し、親セグメントにすべてのレコードを記述する代わりにサブセグメントにメタデータと注釈を記録できます。

SDK のクラスとメソッドのリファレンス ドキュメントについては、AWS X-Ray SDK for Ruby API リファレンスを参照してください。

要件

X-Ray SDK では Ruby 2.3 以降が必要です。また、次のライブラリと互換性があります。

- AWS SDK for Ruby バージョン 3.0 以降
- Rails バージョン 5.1 以降

X-Ray SDK for Ruby の設定

X-Ray SDK for Ruby には、グローバルレコーダーを提供する `XRay.recorder` というクラスがあります。グローバルレコーダーを設定して、受信 HTTP コールのセグメントを作成するミドルウェアをカスタマイズできます。

セクション

- サービスプラグイン
- サンプリングルール

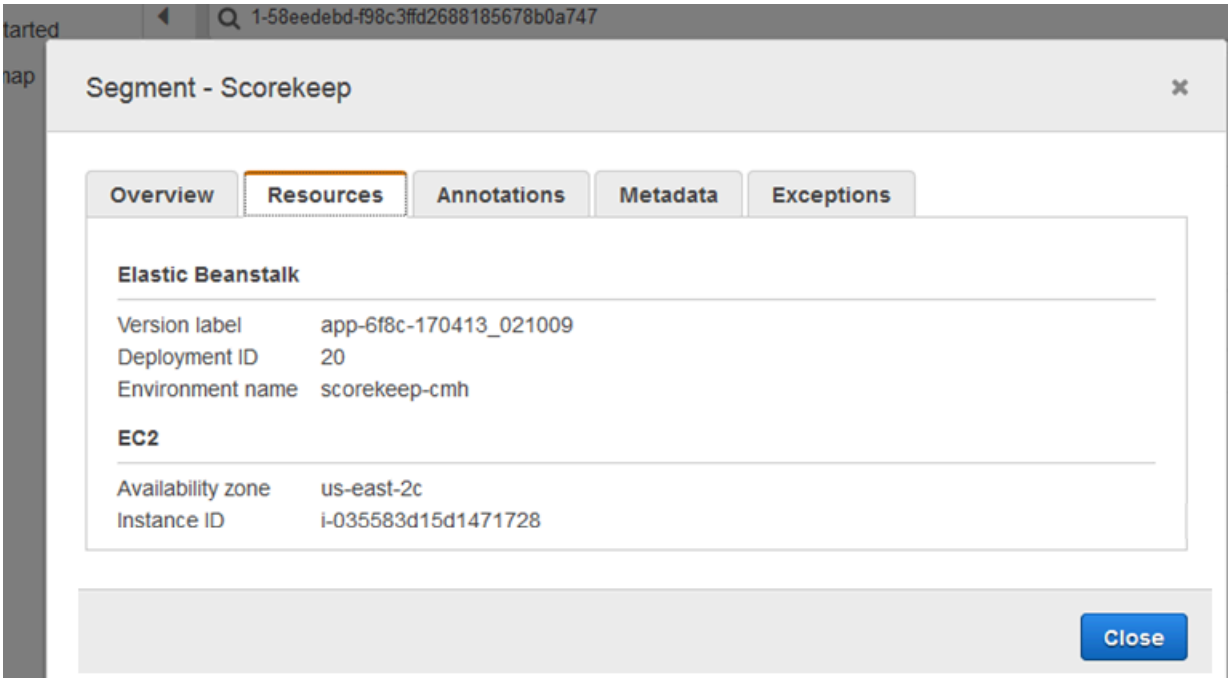
- ログ記録
- コード内のレコーダー設定
- Rails でのレコーダー設定
- 環境変数

サービスプラグイン

`plugins`を使用して、アプリケーションをホストしているサービスに関する情報を記録します。

プラグイン

- Amazon EC2 – `ec2` はインスタンス ID とアベイラビリティーゾーンを追加します。
- Elastic Beanstalk – `elastic_beanstalk` は環境名、バージョンラベル、およびデプロイ ID を追加します。
- Amazon ECS —`ecs`は、コンテナ ID を追加します。



プラグインを使用するには、レコーダーに渡す設定オブジェクトでそのプラグインを指定します。

Example main.rb – プラグインの設定

```
my_plugins = %I[ec2 elastic_beanstalk]
```

```
config = {
  plugins: my_plugins,
  name: 'my app',
}

XRay.recorder.configure(config)
```

また、コードで設定した値よりも優先される環境変数を使用して、レコーダーを設定することもできます。

また、SDK はプラグイン設定を使用して、セグメントの origin フィールドを設定します。これは、アプリケーションを実行する AWS リソースのタイプを示します。複数のプラグインを使用する場合、SDK は次の解決順序を使用してオリジンを決定します： ElasticBeanstalk > EKS > ECS > EC2。

サンプリングルール

SDK は X-Ray コンソールで定義したサンプリングルールを使用し、記録するリクエストを決定します。デフォルトルールでは、最初のリクエストを毎秒トレースし、X-Ray にトレースを送信するすべてのサービスで追加のリクエストの 5% をトレースします。X-Ray コンソールに追加のルールを作成するをクリックして、各アプリケーションで記録されるデータ量をカスタマイズします。

SDK は、定義された順序でカスタムルールを適用します。リクエストが複数のカスタムルールと一致する場合、SDK は最初のルールのみを適用します。

> **Note**
>
> SDK が X-Ray に到達してサンプリングルールを取得できない場合、1 秒ごとに受信された最初のリクエストのデフォルトのローカルルールに戻り、ホストあたりの追加リクエストの 5% に戻ります。これは、ホストがサンプリング API を呼び出す権限を持っていない場合や、SDK によって行われる API 呼び出しの TCP プロキシとして機能する X-Ray デーモンに接続できない場合に発生します。

JSON ドキュメントからサンプリングルールをロードするように SDK を設定することもできます。SDK は、X-Ray サンプリングが利用できない場合のバックアップとしてローカルルールを使用することも、ローカルルールを排他的に使用することもできます。

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
  }
}
```

この例では、1 つのカスタムルールとデフォルトルールを定義します。カスタムルールでは、5 パーセントのサンプリングレートが適用され、/api/move/以下のパスに対してトレースするリクエストの最小数はありません。デフォルトのルールでは、1秒ごとの最初のリクエストおよび追加リクエストの 10 パーセントをトレースします。

ルールをローカルで定義することの欠点は、固定ターゲットが X-Ray サービスによって管理されるのではなく、レコーダーの各インスタンスによって個別に適用されることです。より多くのホストをデプロイすると、固定レートが重複し、記録されるデータ量の制御が難しくなります。

バックアップルールを設定するには、レコーダーに渡す設定オブジェクトで、ドキュメントのハッシュを定義します。

Example main.rb – Backupルールの設定

```
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)
```

サンプリングルールを個別に保存するには、別個のファイルにハッシュを定義し、アプリケーションにそのハッシュをプルするようにファイルに要求します。

Example config/sampling-rules.rb

```
my_sampling_rules =  {
  version: 1,
  default: {
    fixed_target: 1,
    rate: 0.1
  }
}
```

Example main.rb – ファイルからのサンプリングルール

```
require 'aws-xray-sdk'
require 'config/sampling-rules.rb'

config = {
  sampling_rules: my_sampling_rules,
  name: 'my app',
}
XRay.recorder.configure(config)
```

ローカルルールのみを使用するには、サンプリングルールと `LocalSampler` を設定する必要があります。

Example main.rb – ローカルルールサンプリング

```
require 'aws-xray-sdk'
require 'aws-xray-sdk/sampling/local/sampler'

config = {
  sampler: LocalSampler.new,
  name: 'my app',
}
XRay.recorder.configure(config)
```

サンプリングを無効にしてすべての着信リクエストを実装するように、グローバルレコーダーを設定することもできます。

Example main.rb – サンプリングを無効にする

```
require 'aws-xray-sdk'
config = {
  sampling: false,
  name: 'my app',
}
XRay.recorder.configure(config)
```

ログ記録

デフォルトでは、レコーダーは情報レベルのイベントを $stdout に出力します。レコーダーに渡す設定オブジェクトで、ロガーを定義してログ記録をカスタマイズできます。

Example main.rb – ログ記録

```
require 'aws-xray-sdk'
config = {
  logger: my_logger,
  name: 'my app',
}
XRay.recorder.configure(config)
```

デバッグログを使用して問題を識別します。たとえば、「サブセグメントを手動で生成する」場合にサブセグメントが閉じない問題などです。

コード内のレコーダー設定

追加の設定は、XRay.recorder の configure メソッドから利用できます。

- context_missing - 測定されたコードが、セグメントが開いていないときにデータを記録しようとした場合に例外のスローを回避するには、LOG_ERROR に設定します。
- daemon_address – X-Ray デーモンリスナーのホストとポートを設定します。
- name – SDK がセグメントに使用するサービス名を設定します。
- naming_pattern – 動的な命名を使用するようにドメイン名を設定します。
- plugins – プラグインを使用して、アプリケーションの AWS リソースに関する情報を記録します。
- sampling – false に設定してサンプリングを無効にします。

- `sampling_rules` – サンプリングルールを含むハッシュを設定します。

Example main.py – コンテキスト欠落例外を無効にする

```
require 'aws-xray-sdk'
config = {
  context_missing: 'LOG_ERROR'
}

XRay.recorder.configure(config)
```

Rails でのレコーダー設定

Rails フレームワークを使用している場合は、Ruby ファイルの `app_root/initializers` 以下で、グローバルレコーダーのオプションを設定できます。X-Ray SDK は、Rails で使用する追加の設定キーをサポートしています。

- `active_record` – `true`に設定して、Active Record データベーストランザクションのサブセグメントを記録します。

`Rails.application.config.xray` という名前の設定オブジェクトで利用可能な設定を行います。

Example config/initializers/aws_xray.rb

```
Rails.application.config.xray = {
  name: 'my app',
  patch: %I[net_http aws_sdk],
  active_record: true
}
```

環境変数

環境変数を使用して、X-Ray SDK for Ruby を設定できます。SDK は次の変数をサポートしています。

- `AWS_XRAY_TRACING_NAME` – SDK がセグメントに使用するサービス名を設定します。サーブレットフィルタのセグメント命名ルールで設定したサービス名を上書きします。
- `AWS_XRAY_DAEMON_ADDRESS` –X-Ray デーモンリスナーのホストとポートを設定します。デフォルトでは、SDK は、トレースデータをに送信します`127.0.0.1:2000`。この変数は、デーモンを

次のように構成している場合に使用します。別のポートでリッスンするまたは、別のホストで実行されている場合。

- AWS_XRAY_CONTEXT_MISSING – 計測されたコードが、セグメントが開いていないときにデータを記録しようとした場合に例外をスローするには、RUNTIME_ERROR に設定します。

  有効な値

  - RUNTIME_ERROR— ランタイム例外をスローします。
  - LOG_ERROR – エラーをログ記録して続行します (デフォルト)。
  - IGNORE_ERROR – エラーを無視して続行します。

  リクエストが開かれていないときに実行されるスタートアップコード、または新しいスレッドを生成するコードで測定されたクライアントを使用しようとしたときに発生する可能性があるセグメントまたはサブセグメントの欠落に関連するエラー。

環境変数は、コードで設定される値を上書きします。

X-Ray SDK for Ruby ミドルウェアでの受信リクエストのトレーシング

X-Ray SDK を使用して、アプリケーションが Amazon EC2 の EC2 インスタンス AWS Elastic Beanstalk、または Amazon ECS で処理する受信 HTTP リクエストをトレースできます。 Amazon EC2

Rails を使用する場合は、Rails ミドルウェアを使用して、受信 HTTP リクエストを計測します。ミドルウェアをアプリケーションに追加してセグメント名を設定すると、X-Ray SDK for Ruby はサンプリングされたリクエストごとにセグメントを作成します。追加実装で作成されたセグメントは、HTTP リクエストおよびレスポンスに関する情報を提供するリクエストレベルのセグメントのサブセグメントになります。この情報には、時間、メソッド、リクエストの処理などがります。

各セグメントには、サービスマップ内のアプリケーションを識別する名前があります。セグメントの名前は静的に指定することも、受信リクエストのホストヘッダーに基づいて動的に名前を付けるように SDK を設定することもできます。動的ネーミングでは、リクエスト内のドメイン名に基づいてトレースをグループ化でき、名前が予想されるパターンと一致しない場合 (たとえば、ホストヘッダーが偽造されている場合) 、デフォルト名を適用できます。

> **転送されたリクエスト**
>
> ロードバランサーまたは他の仲介者がアプリケーションにリクエストを転送する場合、X-Ray は、クライアントの IP をIP パケットの送信元 IP からではなく、リクエストのX-

Forwarded-Forヘッダーから取得します。転送されたリクエストについて記録されたクライアント IP は偽造される可能性があるため、信頼されるべきではありません。

リクエストが転送されると、それを示す追加フィールドが SDK によってセグメントに設定されます。セグメントのフィールド x_forwarded_for が true に設定されている場合、クライアント IP が HTTP リクエストの X-Forwarded-For ヘッダーから取得されます。

ミドルウェアは、次の情報が含まれる http ブロックを使用して、各受信リクエスト用にセグメントを作成します。

- HTTP メソッド – GET、POST、PUT、DELETE、その他。
- クライアントアドレス – リクエストを送信するクライアントの IP アドレス。
- レスポンスコード – 完了したリクエストの HTTP レスポンスコード。
- タイミング – 開始時間 (リクエストが受信された時間) および終了時間 (レスポンスが送信された時間)。
- ユーザーエージェント — リクエストからのuser-agent
- コンテンツの長さ – レスポンスからの content-length。

Rails ミドルウェアの使用

ミドルウェアを使用するには、gemfile を更新して必要な railtie を含めます。

Example Gemfile - rails

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

ミドルウェアを使用するには、トレースマップ内のアプリケーションを表す名前でレコーダーを設定する必要もあります。

Example config/initializers/aws_xray.rb

```
Rails.application.config.xray = {
  name: 'my app'
}
```

手動によるコードの実装

Rails を使用しない場合は、手動でセグメントを作成します。受信リクエストごとにセグメントを作成したり、パッチが適用された HTTP または AWS SDK クライアントの周囲にセグメントを作成して、レコーダーにコンテキストを提供してサブセグメントを追加したりできます。

```
# 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
```

セグメント命名ルールの設定

AWS X-Ray は、サービス名を使用してアプリケーションを識別し、アプリケーションが使用する他のアプリケーション、データベース、外部 APIs、および AWS リソースと区別します。X-Ray SDK が受信リクエストのセグメントを生成すると、アプリケーションのサービス名がセグメントの名前フィールドに記録されます。

X-Ray SDK では、HTTP リクエストヘッダーのホスト名の後にセグメントの名前を指定できます。ただし、このヘッダーは偽造され、サービスマップに予期しないノードが発生する可能性があります。偽造されたホストヘッダーを持つリクエストによって SDK がセグメントの名前を間違えないようにするには、受信リクエストのデフォルト名を指定する必要があります。

アプリケーションが複数のドメインのリクエストを処理する場合、動的ネーミングストラテジーを使用してセグメント名にこれを反映するように SDK を設定できます。動的ネーミングストラテジーに

より、SDK は予想されるパターンに一致するリクエストにホスト名を使用し、そうでないリクエストにデフォルト名を適用できます。

たとえば、3 つのサブドメイン (www.example.com,api.example.com,およびstatic.example.com) に対してリクエストを処理する単一のアプリケーションがあるとします。動的ネーミングストラテジーをパターン *.example.com で使用して、異なる名前を持つ各サブドメインのセグメントを識別することができます。結果的にはサービスマップ上に 3 つのサービスノードを作成することになります。アプリケーションがパターンと一致しないホスト名のリクエストを受信すると、指定したフォールバック名を持つ 4 番目のノードがサービスマップに表示されます。

すべてのリクエストセグメントに同じ名前を使用するには、前のセクションで示されたように、レコーダーを設定するときにアプリケーションの名前を指定します。

動的命名ルールは、ホスト名と一致するようパターンを定義し、HTTP リクエストのホスト名がパターンと一致しない場合はデフォルトの名前を使用します。セグメントに動的に名前を付けるには、config ハッシュで命名パターンを指定します。

Example main.rb – 動的命名

```
config = {
  naming_pattern: '*mydomain*',
  name: 'my app',
}

XRay.recorder.configure(config)
```

パターン内で任意の文字列に一致させるには「*」を、また、任意の 1 文字に一致させるには「?」を使用することができます。

> **Note**
>
> コードで定義したデフォルトのサービス名は、AWS_XRAY_TRACING_NAME 環境変数で上書きできます。

ダウンストリームコールを実装するためのライブラリへのパッチ適用

ダウンストリームコールを実装するには、X-Ray SDK for Ruby を使用して、アプリケーションが使用するライブラリにパッチを適用します。X-Ray SDK for Ruby では、次のライブラリにパッチを適用できます。

サポートされているライブラリ

- net/http – 実装 HTTP クライアント。
- aws-sdk – 実装AWS SDK for Ruby クライアント。

パッチ適用されたライブラリを使用すると、X-Ray SDK for Ruby はコールのサブセグメントを作成し、リクエストとレスポンスの情報を記録します。SDK ミドルウェアまたは `XRay.recorder.begin_segment` のコールのいずれかから、サブセグメントを作成するために SDK でセグメントを使用できる必要があります。

ライブラリにパッチを適用するには、X-Ray レコーダーに渡す設定オブジェクトでそのパッチを指定します。

Example main.rb – パッチライブラリ

```
require 'aws-xray-sdk'

config = {
  name: 'my app',
  patch: %I[net_http aws_sdk]
}

XRay.recorder.configure(config)
```

X-Ray AWS SDK for Ruby を使用した SDK 呼び出しのトレース

アプリケーションが AWS のサービス を呼び出してデータの保存、キューへの書き込み、または通知の送信を行う場合、X-Ray SDK for Ruby はサブセグメント の呼び出しダウンストリームを追跡します。これらのサービス (Amazon S3 バケットや Amazon SQS キューなど) 内でアクセスするトレースされた AWS のサービス および リソースは、X-Ray コンソールのトレースマップにダウンストリームノードとして表示されます。

X-Ray SDK for Ruby は、ライブラリ にパッチを適用すると、すべての AWS SDK クライアントを自動的に計測します。 aws-sdk個々のクライアントを実装することはできません。

すべてのサービスにおいて、X-Ray コンソールでコールされた API の名前を確認できます。サービスのサブセットの場合、X-Ray SDK はセグメントに情報を追加して、サービスマップでより細かく指定します。

たとえば、実装された DynamoDB クライアントでコールすると、SDK はテーブルをターゲットとするコールのセグメントにテーブル名を追加します。コンソールで、各テーブルはサービスマップ内に個別のノードとして表示され、テーブルをターゲットにしないコール用の汎用の DynamoDB ノードが表示されます。

Example 項目を保存するための DynamoDB に対するコールのサブセグメント

```
{
  "id": "24756640c0d0978a",
  "start_time": 1.480305974194E9,
  "end_time": 1.4803059742E9,
  "name": "DynamoDB",
  "namespace": "aws",
  "http": {
    "response": {
      "content_length": 60,
      "status": 200
    }
  },
  "aws": {
    "table_name": "scorekeep-user",
    "operation": "UpdateItem",
    "request_id": "UBQNSO5AEM8T4FDA4RQDEB94OVTDRVV4K4HIRGVJF66Q9ASUAAJG",
  }
}
```

名前付きリソースにアクセスしたとき、次のサービスをコールすると、サービスマップに追加のノードが作成されます。特定のリソースをターゲットとしないコールでは、サービスの汎用ノードが作成されます。

- Amazon DynamoDB – テーブル名
- Amazon Simple Storage Service – バケットとキー名
- Amazon Simple Queue Service – キュー名

X-Ray SDK でのカスタムサブセグメントの生成

サブセグメントはリクエストを提供するために行われた作業の詳細を記載したトレースのセグメントを拡張します。計測済みクライアント内で呼び出しを行うたびに、X-Ray SDK によってサブセグメントに生成された情報が記録されます。追加のサブセグメントを作成して、他のサブセグメントをグループ化したり、コードセクションのパフォーマンスを測定したり、注釈とメタデータを記録したりできます。

サブセグメントを管理するには、begin_subsegment および end_subsegment メソッドを使用します。

```
subsegment = XRay.recorder.begin_subsegment name: 'annotations', namespace: 'remote'
my_annotations = { id: 12345 }
subsegment.annotations.update my_annotations
XRay.recorder.end_subsegment
```

関数のサブセグメントを作成するには、XRay.recorder.capture へのコールでラップします。

```
XRay.recorder.capture('name_for_subsegment') do |subsegment|
  resp = myfunc() # myfunc is your function
  subsegment.annotations.update k1: 'v1'
  resp
end
```

セグメントまたは別のサブセグメント内にサブセグメントを作成する場合、X-Ray SDK によってその ID が生成され、開始時刻と終了時刻が記録されます。

Example サブセグメントとメタデータ

```
"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"
    }
  },
```

X-Ray SDK for Ruby を使用してセグメントに注釈とメタデータを追加する

アノテーションとメタデータを使用して、リクエスト、環境、またはアプリケーションに関する追加情報を記録できます。X-Ray SDK が作成するセグメントまたは作成するカスタムサブセグメントに、注釈およびメタデータを追加できます。

注釈は文字列、数値、またはブール値を使用したキーと値のペアです。注釈は、フィルタ式用にインデックス付けされます。注釈を使用して、コンソールでトレースをグループ化するため、または`GetTraceSummaries` API を呼び出すときに使用するデータを記録します。

メタデータは、オブジェクトとリストを含む、任意のタイプの値を持つことができるキーバリューのペアですが、フィルタ式に使用するためにインデックスは作成されません。メタデータを使用してトレースに保存する追加のデータを記録しますが、検索で使用する必要はありません。

注釈とメタデータに加えて、セグメントにユーザー ID 文字列を記録することもできます。ユーザー ID はセグメントの個別のフィールドに記録され、検索用にインデックスが作成されます。

セクション

- X-Ray SDK for Ruby で注釈を記録する
- X-Ray SDK for Ruby でメタデータを記録する
- X-Ray SDK for Ruby でユーザー ID を記録する

X-Ray SDK for Ruby で注釈を記録する

注釈を使用して、検索用にインデックスを作成するセグメントまたはサブセグメントに情報を記録します。

注釈の要件

- キー — X-Ray アノテーションのキーには、最大 500 文字の英数字を使用できます。アンダースコア記号 (_) 以外のスペースや記号は使用できません。
- 値 — X-Ray アノテーションの値には、最大 1,000 文字の Unicode 文字を使用できます。
- 注釈の数 — 1 つのトレース 1 つにつき最大 50 個の注釈を使用できます。

注釈を記録するには

1. `xray_recorder` から現在のセグメントまたはサブセグメントの参照を取得します。

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

または

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

2. ハッシュ値を使って update を呼び出します。

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

SDK は、セグメントドキュメントの annotations オブジェクトにキーと値のペアとして、注釈を記録します。同じキーで add_annotations を 2 回呼び出すと、同じセグメントまたはサブセグメントに以前記録された値が上書きされます。

特定の値を持つ注釈のあるトレースを見つけるには、annotations.*key*フィルタ式の キーワードを使用します。

X-Ray SDK for Ruby でメタデータを記録する

メタデータを使用して、検索用にインデックスを作成する必要のないセグメントまたはサブセグメントに情報を記録します。メタデータ値は、文字列、数値、ブール値、または JSON オブジェクトや JSON 配列にシリアル化できる任意のオブジェクトになります。

メタデータを記録するには

1. xray_recorder から現在のセグメントまたはサブセグメントの参照を取得します。

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

または

```
require 'aws-xray-sdk'
```

```
...
document = XRay.recorder.current_subsegment
```

2. 文字列キー、ブール値、数値、文字列値、オブジェクト値、文字列名前空間を使用して metadata を呼び出します。

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

同じキーで metadata を 2 回呼び出すと、同じセグメントまたはサブセグメントに以前記録された値が上書きされます。

X-Ray SDK for Ruby でユーザー ID を記録する

リクエストセグメントにユーザー ID を記録して、リクエストを送信したユーザーを識別します。

ユーザー ID を記録するには

1. xray_recorder から現在のセグメントへの参照を取得します。

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

2. セグメントのユーザーフィールドを、リクエストを送信したユーザーの文字列 ID に設定します。

```
segment.user = 'U12345'
```

コントローラーでユーザーを設定し、アプリケーションがリクエストの処理を開始するとすぐに、ユーザー ID を記録できます。

ユーザー ID のトレースを見つけるには、userフィルタ式で、 キーワードを使用します。

他の AWS X-Ray との統合 AWS のサービス

多くの AWS のサービス は、受信リクエストのサンプリングとヘッダーの追加、X-Ray デーモンの実行、X-Ray へのトレースデータの自動送信など、さまざまなレベルの X-Ray 統合を提供しています。X-Rayとの統合には、次のようなものがあります:

- アクティブ計測 – 受信リクエストをサンプリングして計測します。
- パッシブ計測 – 別のサービスで既にサンプリングされているリクエストを計測します。
- リクエストのトレース – すべての受信リクエストにトレースヘッダーを追加してダウンストリームに伝達します。
- ツール – X-Ray デーモンを実行して X-Ray SDK からセグメントを受信します。

> **Note**
>
> X-Ray SDKsには、 との追加統合用のプラグインが含まれています AWS のサービス。たとえば、X-Ray SDK for Java Elastic Beanstalk プラグイン を使用して、アプリケーション、を実行する Elastic Beanstalk 環境に関する情報 (環境名と ID を含む) を追加できます。

X-Ray と統合 AWS のサービス されている の例をいくつか示します。

- AWS Distro for OpenTelemetry (ADOT) – ADOT を使用すると、エンジニアはアプリケーションを一度計測し、相関メトリクスとトレースを Amazon CloudWatch、Amazon OpenSearch Service AWS X-Ray、Amazon Managed Service for Prometheus などの複数の AWS モニタリングソリューションに送信できます。
- AWS Lambda - すべての runtimes. AWS Lambda adds の受信リクエストのアクティブ計測とパッシブ計測は、2 つのノードをトレースマップに追加します。1 つは AWS Lambda サービス用、もう 1 つは 関数用です。インストルメンテーションを有効にすると、 は X-Ray SDK で使用する Java および Node.js ランタイムでも X-Ray デーモン AWS Lambda を実行します。
- Amazon API Gateway — アクティブおよびパッシブ計測。API Gateway は、サンプリングルールを使用して記録するリクエストを決定し、ゲートウェイステージ用のノードをサービスマップに追加します。
- AWS Elastic Beanstalk – ツール。Elastic Beanstalk には次のプラットフォームで X-Ray デーモンが含まれています。
  - Java SE – 2.3.0 以降の設定

  - Tomcat – 2.4.0 以降の設定
  - Node.js – 3.2.0 以降の設定
  - Windows Server – Windows Server Core を除く、2016 年 12 月 9 日以降にリリースされたすべての設定

  Elastic Beanstalk コンソールを使用するか、`aws:elasticbeanstalk:xray` 名前空間で `XRayEnabled` オプションを使用して、これらのプラットフォームでデーモンを実行するように Elastic Beanstalk を設定できます。

- Elastic Load Balancing — Application Load Balancerでトレースを要求します。Application Load Balancerはトレース ID をリクエストヘッダーに追加してからターゲットグループに送信します。
- Amazon EventBridge - パッシブ計測。イベント EventBridgeを発行するサービスが X-Ray SDK を使用して計測されている場合、イベントターゲットはトレースヘッダーを受け取り、元のトレース ID を引き続き伝達できます。
- Amazon Simple Notification Service — パッシブ計測。Amazon SNS パブリッシャーが X-Ray SDK クライアントを使用してクライアントをトレースする場合、サブスクライバーはトレースヘッダーを取得し、同じトレース ID を使用して、パブリッシャーからの元のトレースを継続して伝達できます。
- Amazon Simple Queue Service — パッシブ計測。サービスが X-Ray SDK を使用してリクエストをトレースする場合、Amazon SQS はトレースヘッダーを送信し、整合性のあるトレース ID を持つコンシューマーに、送信者から元のトレースを伝達し続けます。

次のトピックから選択して、統合された の完全なセットを確認します AWS のサービス。

トピック

- AWS Distro for OpenTelemetry と AWS X-Ray
- の Amazon API Gateway アクティブトレースのサポート AWS X-Ray
- Amazon EC2 と AWS App Mesh
- AWS App Runner と X-Ray
- AWS AppSyncおよびAWS X-Ray
- を使用した X-Ray API コールのログ記録 AWS CloudTrail
- CloudWatch と X-Ray の統合
- X-Rayの暗号化設定の変更を追跡AWS Config
- Amazon Elastic Compute CloudおよびAWS X-Ray

- AWS Elastic BeanstalkおよびAWS X-Ray
- Elastic Load Balancing と AWS X-Ray
- Amazon EventBridge と AWS X-Ray
- AWS Lambda および AWS X-Ray
- Amazon SNS と AWS X-Ray
- AWS Step Functions および AWS X-Ray
- Amazon SQS と AWS X-Ray
- Amazon S3 と AWS X-Ray

AWS Distro for OpenTelemetry と AWS X-Ray

AWS Distro for OpenTelemetry (ADOT) を使用してメトリクスとトレースを収集し、AWS X-Ray および Amazon CloudWatch、Amazon OpenSearch Service、Amazon Managed Service for Prometheus などの他のモニタリングソリューションに送信します。

AWS Distro for OpenTelemetry

The AWS Distro for OpenTelemetry（ADOT）は、クラウドネイティブコンピューティング財団（CNCF）のOpenTelemetryプロジェクトに基づいたAWS ディストリビューションです。OpenTelemetry は、分散トレースとメトリクスを収集するためのオープンソース API、ライブラリ、およびエージェントの 単一セットを提供します。このツールキットは、SDK、自動計測エージェント、およびコレクタを含むアップストリームの OpenTelemetry コンポーネントのディストリビューションであり、によってテスト、最適化、保護、およびサポートされます。AWS。

ADOT を使用すると、エンジニアはアプリケーションを一度計測し、相関のあるメトリクスとトレースを Amazon CloudWatch、AWS X-Ray、Amazon OpenSearch Service、Amazon Managed Service for Prometheus などの複数の AWS モニタリングソリューションに送信できます。

ADOT は、トレースやメトリクスを X-Ray などのモニタリングソリューションへ簡単に送信できるようにするため、多数の AWS のサービス と統合されています。ADOT と統合されたサービスの例として、次のようなものがあります。

- AWS Lambda – ADOT 用の AWS マネージド型 Lambda レイヤーは、Lambda 関数を自動的に計測し、OpenTelemetry を AWS Lambda と X-Ray のすぐに使える設定とともにセットアップの簡単なレイヤーにパッケージ化することで、プラグアンドプレイのユーザーエクスペリエンスを提供します。ユーザーは、コードを変更せずに Lambda 関数の OpenTelemetry を有効または無効にできます。詳細については、「AWS Distro for OpenTelemetry Lambda」を参照してください。

- Amazon Elastic Container Service (ECS) – AWS Distro for OpenTelemetry Collector を使用して Amazon ECS アプリケーションからメトリクスとトレースを収集し、X-Ray やその他のモニタリングソリューションに送信します。詳細については、Amazon ECS デベロッパーガイドの「アプリケーショントレースデータの収集」を参照してください。
- AWS App Runner – App Runner は、AWS Distro for OpenTelemetry (ADOT) を使用した X-Ray へのトレースの送信をサポートしています。ADOT SDK を使用してコンテナ化されたアプリケーションのトレースデータを収集し、X-Ray を使用して計測したアプリケーションを分析し、インサイトを得ます。詳細については、「AWS App Runner と X-Ray」を参照してください。

追加の AWS のサービス との統合など AWS Distro for OpenTelemetry の詳細については、「AWS Distro for OpenTelemetry Documentation」を参照してください。

AWS Distro for OpenTelemetry と X-Ray でアプリケーションを計測する方法の詳細については、「AWS Distro for OpenTelemetry でアプリケーションを計測する」を参照してください。

の Amazon API Gateway アクティブトレースのサポート AWS X-Ray

X-Rayを使用して、ユーザーリクエストがAmazon API Gateway APIsを経由し、基礎となるサービスへの流れをトレースして分析することができます。API Gatewayは、すべてのAPI Gateway評価項目タイプでX - Rayトレースがサポートしています：地域、エッジの最適化、プライベートです。X-Ray は、X-Ray AWS リージョン が利用可能なすべての で Amazon API Gateway で使用できます。詳細については、『Amazon API Gateway開発者ガイド』 AWS X-Rayの「API Gateway APIの実行の追跡」を参照してください。

> **Note**
>
> X-Rayは、 API Gatewayを介したREST APIの追跡のみをサポートしています。

Amazon API Gateway は、 のアクティブトレースサポートを提供します AWS X-Ray。APIステージの追跡機能を有効にすると、受信リクエストをサンプリングし、X-Rayにトレースを送信することができます。

APIステージで追跡機能を有効にするには

1. API Gateway(https://console.aws.amazon.com/apigateway)コンソールを開きます。

2. APIを選択します。
3. ステージを選択します。
4. 「ログ/トレース」タブで、以下を選択します、「X-Rayトレースを有効にする」を選択し、変更の保存。
5. 左側のナビゲーションパネルで、「リソース」を選択します。
6. 新しい設定でAPIを再配置するには、[Actions]ドロップダウン方式を選択し[Deploy API]の順に選択します。

API Gatewayは、X-Rayコンソールで定義したサンプリングルールを使用し、記録するリクエストを決定します。APIのみに適用されるルール、または特定のヘッダーを含むリクエストにのみ適用されるルールを作成することができます。API Gatewayは、ステージおよびリクエストの詳細と共に、セグメント上の属性にヘッダーを記録します。詳細については、「サンプリングルールを設定する」を参照してください。

> **Note**
>
> API Gateway HTTP 統合 で REST APIs をトレースする場合、各セグメントのサービス名は API Gateway から HTTP 統合エンドポイントへのリクエスト URL パスに設定され、一意の各 URL パスの X-Ray トレースマップ上のサービスノードになります。URL パスの数が多いと、トレースマップが 10,000 ノードの制限を超える可能性があり、エラーが発生する可能性があります。
> API Gateway によって作成されるサービスノードの数を最小限に抑えるには、パラメータを URL クエリ文字列内、またはリクエスト本文に入れて POST 経由で渡すことを検討してください。いずれの方法でも、パラメータが URL パスに含まれることはないため、個別の URL パスやサービスノードの数が少なくなる可能性があります。

API Gatewayは、HTTP (すべての受信ハイパーテキスト転送プロトコル) リクエストに対し、まだトレースヘッダーを持たない受信リクエストにトレースヘッダーを追加します。

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

X-Ray トレース ID 形式

X-Ray `trace_id` は、ハイフンで区切られた 3 つの数字で構成されています。例えば、`1-58406520-a006649127e371903a2de979` と指定します。これには、以下のものが含まれます：

- バージョン番号。 1
- 8 桁の 16 進数 を使用した Unix エポックタイムでの元のリクエストの時刻。

  例えば、2016 年 12 月 1 日午前 10:00 PST のエポックタイムは1480615200秒または 16 進数58406520です。
- 24 桁の 16 進数 でトレースにグローバルに一意の 96 ビット識別子。

アクティブなトレースが無効になっている場合でも、リクエストがサンプリングされ、かつトレースが開始されているサービスからのリクエストが送信された場合は、ステージによりセグメントが記録されます。たとえば、搭載されたウェブアプリケーションがHTTPクライアントを使用してAPI Gateway APIを呼び出すことができます。X-Ray SDKを使用してHTTPクライアントを導入すると、サンプリング判定を含む発信リクエストにトレースヘッダーが追加されます。API Gatewayは、トレースヘッダーを読み取り、サンプリングされたリクエストのセグメントを作成します。

API Gateway を使用して API の Java SDK を生成する場合、 SDK クライアントを手動で計測するのと同じ方法で、クライアントビルダーでリクエストハンドラーを追加することで AWS SDK クライアントを計測できます。手順については、「X-Ray AWS SDK for Java を使用した SDK 呼び出しのトレース」を参照してください。

Amazon EC2 と AWS App Mesh

AWS X-Ray は と統合AWS App Meshして、マイクロサービスの Envoy プロキシを管理します。App Meshでは、同じタスクまたはポッドのコンテナ内で実行されているX-Rayデーモンにトレースデータを送信するように設定できるEnvoyバージョンを提供しています。X-Rayは、以下のApp Mesh対応サービスでのトレースをサポートしています:

- Amazon Elastic Container Service (Amazon ECS)
- Amazon Elastic Kubernetes Service (Amazon EKS)
- Amazon Elastic Compute Cloud (Amazon EC2)

App MeshでX-Rayトレースを有効にする方法については、以下の説明を参照にしてください。

Envoy proxyがX-Rayにデータを送信するように設定するには、`ENABLE_ENVOY_XRAY_TRACING` コンテナの定義で環境変数を設定します。

> **Note**
>
> App MeshバージョンのEnvoyは現在、サンプリングのルール設定に基づいてトレースを送信しません。代わりに、Envoyバージョン1.16.3 以降では5%の固定サンプリングレートを使用し、Envoyバージョン1.16.3以前では50%のサンプリングレートを使用します。

Example Amazon ECSのためのEnvoyコンテナの定義

```
{
      "name": "envoy",
      "image": "public.ecr.aws/appmesh/aws-appmesh-envoy:envoy-version",
      "essential": true,
      "environment": [
        {
          "name": "APPMESH_VIRTUAL_NODE_NAME",
          "value": "mesh/myMesh/virtualNode/myNode"
        },
        {
          "name": "ENABLE_ENVOY_XRAY_TRACING",
          "value": "1"
        }
      ],
      "healthCheck": {
        "command": [
          "CMD-SHELL",
          "curl -s http://localhost:9901/server_info | cut -d' ' -f3 | grep -q live"
        ],
        "startPeriod": 10,
        "interval": 5,
        "timeout": 2,
        "retries": 3
      }
```

> **Note**
>
> 使用可能なEnvoy リージョンのアドレスの詳細については、「ユーザーガイド」のの「 AWS App Mesh Envoyイメージ」を参照してください。

X-Rayデーモンをコンテナ内で実行する方法について参照してくださいAmazon ECS での X-Ray デーモンの実行。サービスメッシュ、マイクロサービス、Envoy プロキシ、および X-Ray デーモンを含むサンプルアプリケーションの場合は、App Mesh サンプル GitHubリポジトリ でcolorappサンプルをデプロイします。

詳細はこちら

- AWS App Meshの開始方法

- AWS App Mesh および Amazon ECS の開始方法

AWS App Runner と X-Ray

AWS App Runner は、AWS クラウド で、ソースコードまたはコンテナイメージから、スケーラブルでセキュアなウェブアプリケーションに直接デプロイする、迅速、シンプル、費用対効果の高い方法を提供する AWS のサービス です。新しいテクノロジーを学習したり、使用するコンピューティングサービスを決定したり、AWS リソースのプロビジョニングと構成方法を知ったりする必要はありません。詳細については、「AWS App Runner とは」を参照してください。

AWS App Runner は AWS Distro for OpenTelemetry (ADOT) との統合により、トレースを X-Ray に送信します。ADOT SDK を使用してコンテナ化されたアプリケーションのトレースデータを収集し、X-Ray を使用して計測したアプリケーションを分析し、インサイトを得ます。詳細については、「X-Ray を使用した App Runner アプリケーションのトレース」を参照してください。

AWS AppSyncおよびAWS X-Ray

AppSyncのリクエストを有効にしたりトレースするこAWSとができます。詳細については、AWS「X-Rayのためのトレース」を参照してください。

X-RayトレースがAWSAppSync APIAWSに対して有効になっている場合、Identity and Access Managementのサービスにリンクした役割で適切なアクセス許可が付与され、自動的にアカウント内に作成されます。これにより、AWSAppSyncは安全な方法でX-Rayにトレースを送信することができます。

を使用した X-Ray API コールのログ記録 AWS CloudTrail

AWS X-Ray は、ユーザーAWS CloudTrail、ロール、または によって実行されたアクションを記録するサービスである と統合されています AWS のサービス。 は、X-Ray のすべての API コールをイベントとして CloudTrail キャプチャします。キャプチャされた呼び出しには、X-Ray コンソールからの呼び出しと、X-Ray API オペレーションへのコード呼び出しが含まれます。で収集された情報を使用して CloudTrail、X-Ray に対するリクエスト、リクエスト元の IP アドレス、リクエスト日時などの詳細を確認できます。

各イベントまたはログエントリには、誰がリクエストを生成したかという情報が含まれます。アイデンティティ情報は、以下を判別するために役立ちます。

- ルートユーザーまたはユーザー認証情報のどちらを使用してリクエストが送信されたか

- リクエストが IAM Identity Center ユーザーに代わって行われたかどうか。
- リクエストがロールまたはフェデレーションユーザーのテンポラリなセキュリティ認証情報を使用して行われたかどうか。
- リクエストが、別の AWS のサービスによって送信されたかどうか。

CloudTrail アカウント AWS アカウント を作成すると、 が でアクティブになり、 CloudTrail イベント履歴 に自動的にアクセスできます。 CloudTrail イベント履歴は、 に記録された過去 90 日間の管理イベントの表示、検索、ダウンロード、およびイミュータブルな記録を提供します AWS リージョン。詳細については、「 ユーザーガイド」の CloudTrail 「イベント履歴の使用AWS CloudTrail 」を参照してください。イベント履歴の表示には料金はかかりません CloudTrail。

AWS アカウント 過去 90 日間のイベントを継続的に記録するには、証跡または CloudTrail Lake イベントデータストアを作成します。

CloudTrail 証跡

証跡により、 はログファイル CloudTrail を Amazon S3 バケットに配信できます。を使用して作成された証跡はすべてマルチリージョン AWS Management Console です。 AWS CLIを使用する際は、単一リージョンまたは複数リージョンの証跡を作成できます。 AWS リージョン アカウントのすべての でアクティビティをキャプチャするため、マルチリージョンの証跡を作成することをお勧めします。単一リージョンの証跡を作成する場合、証跡の AWS リージョンに記録されたイベントのみを表示できます。証跡の詳細については、「AWS CloudTrail ユーザーガイド」の「AWS アカウントの証跡の作成」および「組織の証跡の作成」を参照してください。

証跡を作成 CloudTrail することで、 から進行中の管理イベントのコピーを 1 つ無料で Amazon S3 バケットに配信できますが、Amazon S3 ストレージ料金が発生します。 CloudTrail 料金の詳細については、AWS CloudTrail 「 の料金」を参照してください。Amazon S3 の料金に関する詳細については、「Amazon S3 の料金」を参照してください。

CloudTrail Lake イベントデータストア

CloudTrail Lake では、イベントに対して SQL ベースのクエリを実行できます。 CloudTrail Lake は、既存のイベントを行ベースの JSON 形式で Apache ORC 形式に変換します。ORC は、データを高速に取得するために最適化された単票ストレージ形式です。イベントはイベントデータストアに集約されます。イベントデータストアは、高度なイベントセレクタを適用することによって選択する条件に基いた、イベントのイミュータブルなコレクションです。どのイベントが存続し、クエリに使用できるかは、イベントデータストアに適用するセレクタが制御します。CloudTrail Lake の詳細については、「 ユーザーガイド」の AWS CloudTrail 「Lake の使用AWS CloudTrail 」を参照してください。

CloudTrail Lake イベントデータストアとクエリにはコストが発生します。イベントデータストアを作成する際に、イベントデータストアに使用する料金オプションを選択します。料金オプションによって、イベントの取り込みと保存にかかる料金、および、そのイベントデータストアのデフォルトと最長の保持期間が決まります。 CloudTrail 料金の詳細については、AWS CloudTrail「 の料金」を参照してください。

トピック

- での X-Ray 管理イベント CloudTrail
- での X-Ray データイベント CloudTrail
- X-Ray イベントの例

での X-Ray 管理イベント CloudTrail

AWS X-Ray は と統合 AWS CloudTrail して、ユーザー、ロール、または によって X-Ray AWS のサービス で実行された API アクションを記録します。 CloudTrail を使用して X-Ray API リクエストをリアルタイムでモニタリングし、Amazon S3、Amazon CloudWatch Logs、Amazon CloudWatch Events にログを保存できます。X-Ray では、以下のアクションをイベントとして CloudTrail ログファイルに記録できます。

サポートされているAPIアクション

- PutEncryptionConfig
- GetEncryptionConfig
- CreateGroup
- UpdateGroup
- DeleteGroup
- GetGroup
- GetGroups
- GetInsight
- GetInsightEvents
- GetInsightImpactGraph
- GetInsightSummaries
- GetSamplingStatisticSummaries

での X-Ray データイベント CloudTrail

データイベントは、リソース (セグメントドキュメントを X-Ray にアップロードPutTraceSegmentsする など) で実行されたリソースオペレーションに関する情報を提供します。

これらのイベントは、データプレーンオペレーションとも呼ばれます。データイベントは、多くの場合、高ボリュームのアクティビティです。デフォルトでは、 CloudTrail はデータイベントを記録しません。 CloudTrail イベント履歴にはデータイベントは記録されません。

追加の変更がイベントデータに適用されます。 CloudTrail 料金の詳細については、AWS CloudTrail「 の料金」を参照してください。

CloudTrail コンソール、、または CloudTrail API オペレーションを使用して AWS CLI、X-Ray リソースタイプのデータイベントをログに記録できます。データイベントをログに記録する方法の詳細については、「 AWS CloudTrail ユーザーガイド」の「 AWS Management Consoleを使用したデータイベントのログ記録」および「AWS Command Line Interfaceを使用したデータイベントのログ記録」を参照してください。

次の表に、データイベントをログに記録できる X-Ray リソースタイプを示します。データイベントタイプ (コンソール) 列には、 CloudTrail コンソールのデータイベントタイプリストから選択する値が表示されます。resources.type 値列には、 AWS CLI または CloudTrail APIs を使用して高度なイベントセレクタを設定するときに指定する `resources.type`値が表示されます。列にログ記録された Data APIs CloudTrail には、リソースタイプ CloudTrail について にログ記録された API コールが表示されます。

データイベントタイプ (コンソール)	resources.type 値	にログ記録APIs CloudTrail
X-Ray トレース	`AWS::XRay::Trace`	• PutTraceSegments • GetTraceSummaries • GetTraceGraph • GetServiceGraph • BatchGetTraces • GetTimeSeriesServiceStatistics • PutTelemetryRecords • GetSamplingTargets

高度なイベントセレクタを設定して、 フィールドeventNameと readOnlyフィールドでフィルタリングし、自分にとって重要なイベントのみをログに記録できます。ただし、X-Ray トレースには ARNs がないため、resources.ARNフィールドセレクタを追加してイベントを選択することはできません。オブジェクトの詳細については、「AWS CloudTrail API リファレンス」の「AdvancedFieldSelector」を参照してください。以下は、 put-event-selectors AWS CLI コマンドを実行して証 CloudTrail 跡のデータイベントを記録する方法の例です。でコマンドを実行するか、証跡が作成されたリージョンを指定する必要があります。それ以外の場合、オペレーションはInvalidHomeRegionException例外を返します。

```
aws cloudtrail put-event-selectors --trail-name myTrail --advanced-event-selectors \
'{
   "AdvancedEventSelectors": [
      {
         "FieldSelectors": [
            { "Field": "eventCategory", "Equals": ["Data"] },
            { "Field": "resources.type", "Equals": ["AWS::XRay::Trace"] },
            { "Field": "eventName", "Equals":
 ["PutTraceSegments","GetSamplingTargets"] }
         ],
         "Name": "Log X-Ray PutTraceSegments and GetSamplingTargets data events"
      }
   ]
}'
```

X-Ray イベントの例

管理イベントの例、 **GetEncryptionConfig**

以下は、 の X-Ray GetEncryptionConfigログエントリの例です CloudTrail。

Example

```
{
    "eventVersion"=>"1.05",
    "userIdentity"=>{
        "type"=>"AssumedRole",
        "principalId"=>"AROAJVHBZWD3DN6CI2MHM:MyName",
        "arn"=>"arn:aws:sts::123456789012:assumed-role/MyRole/MyName",
        "accountId"=>"123456789012",
        "accessKeyId"=>"AKIAIOSFODNN7EXAMPLE",
        "sessionContext"=>{
            "attributes"=>{
```

```
                "mfaAuthenticated"=>"false",
                "creationDate"=>"2023-7-01T00:24:36Z"
            },
            "sessionIssuer"=>{
                "type"=>"Role",
                "principalId"=>"AROAJVHBZWD3DN6CI2MHM",
                "arn"=>"arn:aws:iam::123456789012:role/MyRole",
                "accountId"=>"123456789012",
                "userName"=>"MyRole"
            }
        }
    },
    "eventTime"=>"2023-7-01T00:24:36Z",
    "eventSource"=>"xray.amazonaws.com",
    "eventName"=>"GetEncryptionConfig",
    "awsRegion"=>"us-east-2",
    "sourceIPAddress"=>"33.255.33.255",
    "userAgent"=>"aws-sdk-ruby2/2.11.19 ruby/2.3.1 x86_64-linux",
    "requestParameters"=>nil,
    "responseElements"=>nil,
    "requestID"=>"3fda699a-32e7-4c20-37af-edc2be5acbdb",
    "eventID"=>"039c3d45-6baa-11e3-2f3e-e5a036343c9f",
    "eventType"=>"AwsApiCall",
    "recipientAccountId"=>"123456789012"
}
```

データイベントの例、 **PutTraceSegments**

以下は、 の X-Ray PutTraceSegments データイベントログエントリの例です CloudTrail。

Example

```
{
  "eventVersion": "1.09",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AROAWYXPW54Y4NEXAMPLE:i-0dzz2ac111c83zz0z",
    "arn": "arn:aws:sts::012345678910:assumed-role/my-service-role/
i-0dzz2ac111c83zz0z",
    "accountId": "012345678910",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
```

```
        "type": "Role",
        "principalId": "AROAWYXPW54Y4NEXAMPLE",
        "arn": "arn:aws:iam::012345678910:role/service-role/my-service-role",
        "accountId": "012345678910",
        "userName": "my-service-role"
      },
      "attributes": {
        "creationDate": "2024-01-22T17:34:11Z",
        "mfaAuthenticated": "false"
      },
      "ec2RoleDelivery": "2.0"
    }
  },
  "eventTime": "2024-01-22T18:22:05Z",
  "eventSource": "xray.amazonaws.com",
  "eventName": "PutTraceSegments",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "198.51.100.0",
  "userAgent": "aws-sdk-ruby3/3.190.0 md/internal ua/2.0 api/xray#1.0.0 os/linux md/
x86_64 lang/ruby#2.7.8 md/2.7.8 cfg/retry-mode#legacy",
  "requestParameters": {
    "traceSegmentDocuments": [
      "trace_id:1-00zzz24z-EXAMPLE4f4e41754c77d0000",
      "trace_id:1-00zzz24z-EXAMPLE4f4e41754c77d0000",
      "trace_id:1-00zzz24z-EXAMPLE4f4e41754c77d0001",
      "trace_id:1-00zzz24z-EXAMPLE4f4e41754c77d0002"
    ]
  },
  "responseElements": {
    "unprocessedTraceSegments": []
  },
  "requestID": "5zzzzz64-acbd-46ff-z544-451a3ebcb2f8",
  "eventID": "4zz51z7z-77f9-44zz-9bd7-6c8327740f2e",
  "readOnly": false,
  "resources": [
    {
      "type": "AWS::XRay::Trace"
    }
  ],
  "eventType": "AwsApiCall",
  "managementEvent": false,
  "recipientAccountId": "012345678910",
  "eventCategory": "Data",
  "tlsDetails": {
```

```
    "tlsVersion": "TLSv1.2",
    "cipherSuite": "ZZZZZ-RSA-AAA128-GCM-SHA256",
    "clientProvidedHostHeader": "example.us-west-2.xray.cloudwatch.aws.dev"
  }
}
```

CloudWatch と X-Ray の統合

AWS X-Ray は CloudWatch Application Signals 、 CloudWatch RUM、および CloudWatch Synthetics と統合されているため、アプリケーションの状態を簡単にモニタリングできます。Application Signals のアプリケーションを有効にして、サービス、クライアントページ、Synthetics Canary、およびサービスの依存関係の運用状態をモニタリングおよびトラブルシューティングします。

CloudWatch メトリクス、ログ、X-Ray トレースを関連付けることで、X-Ray トレースマップはサービスの end-to-end ビューを提供し、パフォーマンスのボトルネックを迅速に特定し、影響を受けるユーザーを特定するのに役立ちます。

CloudWatch RUM を使用すると、実際のユーザーモニタリングを実行して、実際のユーザーセッションからウェブアプリケーションのパフォーマンスに関するクライアント側のデータをほぼリアルタイムで収集して表示できます。 AWS X-Ray と CloudWatch RUM を使用すると、ダウンストリームの AWS マネージドサービスを通じてアプリケーションのエンドユーザーから開始するリクエストパスを分析およびデバッグできます。これにより、エンドユーザーに影響を与えるレイテンシーの傾向や エラーを特定できます。

トピック

- CloudWatch RUM と AWS X-Ray
- X-Ray を使用した CloudWatch Synthetics Canary のデバッグ

CloudWatch RUM と AWS X-Ray

Amazon CloudWatch RUM を使用すると、実際のユーザーモニタリングを実行して、実際のユーザーセッションからウェブアプリケーションのパフォーマンスに関するクライアント側のデータをほぼリアルタイムで収集して表示できます。 AWS X-Ray と CloudWatch RUM を使用すると、ダウンストリームの AWS マネージドサービスを通じてアプリケーションのエンドユーザーから開始するリクエストパスを分析およびデバッグできます。これにより、エンドユーザーに影響を与えるレイテンシーの傾向や エラーを特定できます。

ユーザーセッションの X-Ray トレースを有効にすると、 CloudWatch RUM は許可された HTTP リクエストに X-Ray トレースヘッダーを追加し、許可された HTTP リクエストの X-Ray セグメントを記録します。その後、X-Ray トレースマップを含む X-Ray および CloudWatch コンソールで、これらのユーザーセッションからのトレースとセグメントを確認できます。

> **Note**
>
> CloudWatch RUM は X-Ray サンプリングルールと統合されません。代わりに、 CloudWatch RUM を使用するようにアプリケーションを設定するときにサンプリングパーセンテージを選択します。 CloudWatch RUM から送信されたトレースには、追加コストが発生する可能性があります。詳細については、AWS X-Ray の料金を参照してください。

デフォルトでは、 CloudWatch RUM から送信されたクライアント側のトレースはサーバー側のトレースに接続されません。クライアント側のトレースをサーバー側のトレースに接続するには、これらの HTTP リクエストに X-Ray トレースヘッダーを追加するように CloudWatch RUM ウェブクライアントを設定します。

> **Warning**
>
> HTTP リクエストに X-Ray トレースヘッダーを追加するように CloudWatch RUM ウェブクライアントを設定すると、Cross-Origin Resource Sharing (CORS) が失敗する可能性があります。これを回避するには、ダウンストリームサービスの CORS 設定で、許可されるヘッダーのリストに `X-Amzn-Trace-Id` HTTP ヘッダーを追加します。API Gateway をダウンストリームとして使用している場合は、「REST API リソースの CORS を有効にする」を参照してください。本番環境でクライアント側の X-Ray トレースヘッダーの追加を行う前に、アプリケーションのテストを実施することを強くお勧めします。詳細については、CloudWatch RUM ウェブクライアントのドキュメント「」を参照してください。

での実際のユーザーモニタリングの詳細については CloudWatch、 CloudWatch 「RUM の使用」を参照してください。X-Ray でのユーザーセッションのトレースなど、 CloudWatch RUM を使用するようにアプリケーションを設定するには、 CloudWatch 「RUM を使用するようにアプリケーションをセットアップする」を参照してください。

X-Ray を使用した CloudWatch Synthetics Canary のデバッグ

CloudWatch Synthetics は、1 日 24 時間、1 分に 1 回実行されるスクリプト化された Canary を使用してエンドポイントと APIs をモニタリングできるフルマネージドサービスです。

以下の変更を確認して、canaryスクリプトをカスタマイズできます：

- 可用性
- レイテンシー
- トランザクション
- リンク切れまたはデットリンク
- S tep-by-step タスクの完了
- ページロードエラー
- UI アセットのロードレイテンシー
- 複雑なウィザードフロー
- アプリケーションのチェックアウトフロー

カナリアは、お客様と同じルートをたどり、同じアクションと動作を実行して、お客様の満足体験を継続的に検証します。

Syntheticsテストの設定の詳細については、「Syntheticsを使用してカナリアを作成および管理する」を参照してください。

以下の例では、Syntheticsのカナリアで発生する問題をデバッグするための一般的な使用例を示しています。各例は、トレースマップまたは X-Ray Analytics コンソールを使用してデバッグするための主要な戦略を示しています。

トレースマップの読み取りおよび操作方法の詳細については、「サービスマップの表示」を参照してください。

X-Rey Analyticsコンソールの読み方および操作方法の詳細については、「 AWS X-Ray Analyticsコンソールの操作」を参照してください。

トピック

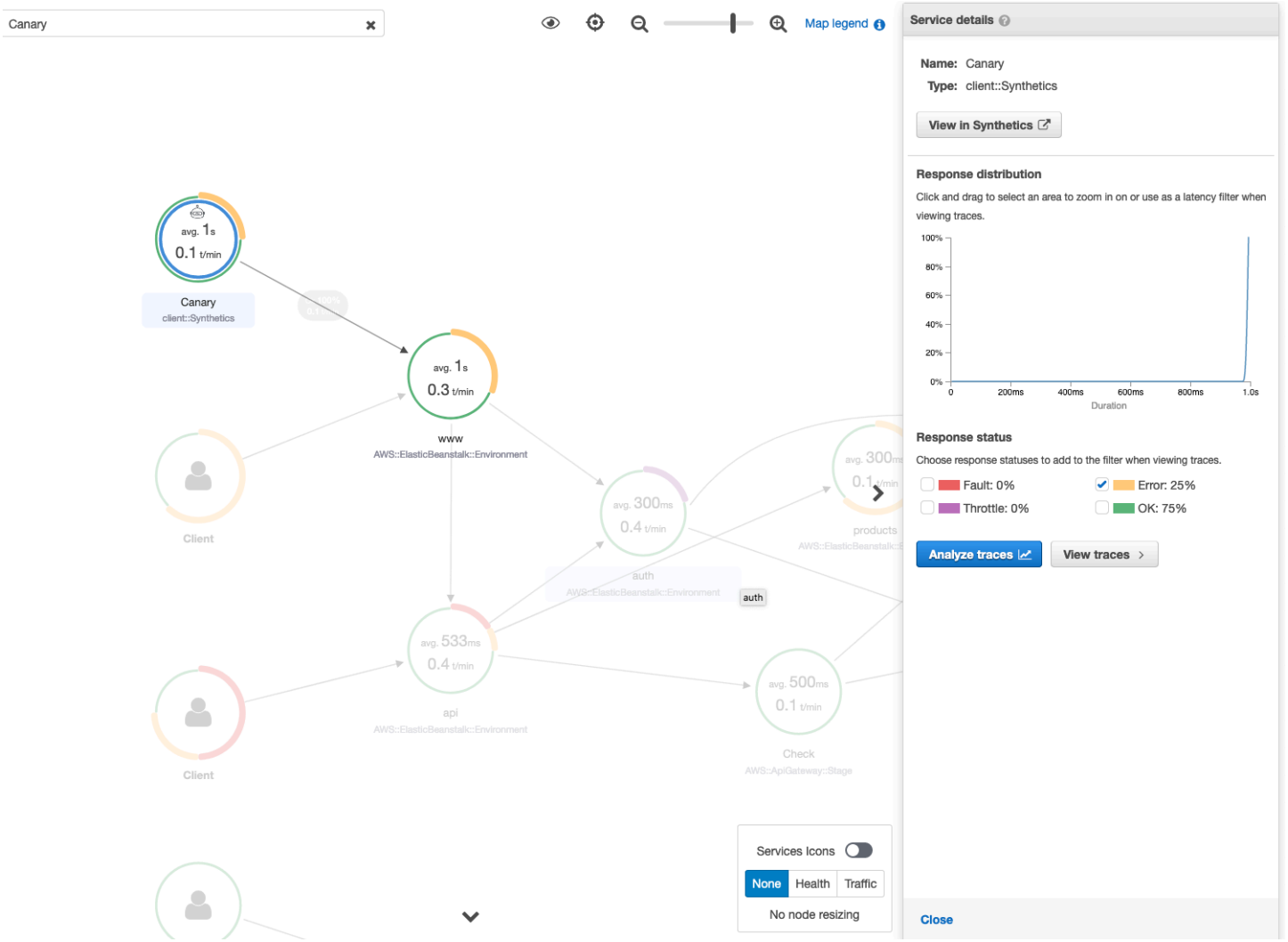
- トレースマップでエラーレポートが増加した Canary を表示する
- 個々のトレースのトレース詳細マップを使用して、各リクエストを詳細に表示する

- アップストリームおよびダウンストリームサービスで継続的に発生している障害の根本原因を特定する
- パフォーマンスのボトルネックとトレンドを特定する
- 変更前と変更後で待ち時間およびエラー・障害率を比較する
- すべてのAPIとURLに必要なcanaryの受信可能範囲エリアを特定する
- グループを使用してSyntheticsテストに焦点を合わせる

トレースマップでエラーレポートが増加した Canary を表示する

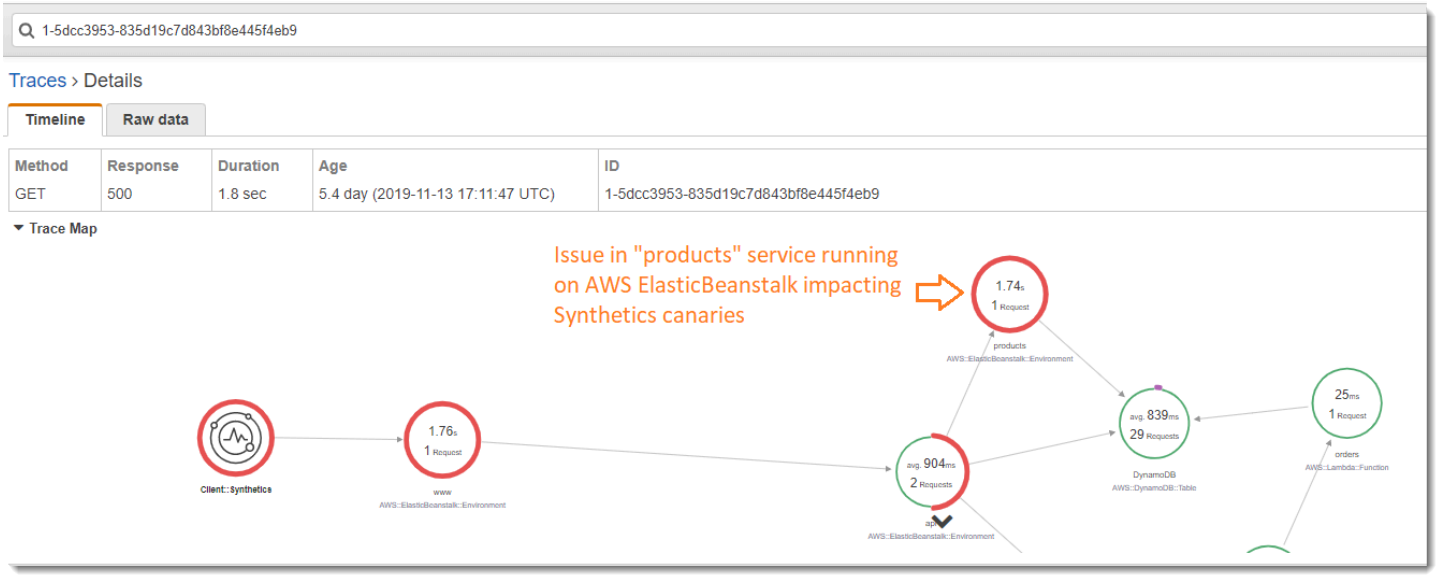
X-Ray トレースマップ内のエラー、障害、スロットリング率、または応答時間が遅い Canary を確認するには、 `Client::Synthetic`フィルターを使用して Synthetics Canary クライアントノードを強調表示できます。詳細については、「フィルター式を使用する」を参照してください。ノードを選択すると、リクエスト全体の応答時間の分布が表示されます。2 つのノード間で エッジを選択すると、その接続を通過したリクエストの詳細が表示されます。トレースマップで、関連するダウンストリームサービスの「リモート」推定ノードを表示することもできます。

Synthetics ノードを選択すると、サイドパネルに「Synthetics で表示」ボタンが表示され、Synthetics コンソールにリダイレクトされ、Canary の詳細を確認できます。



個々のトレースのトレース詳細マップを使用して、各リクエストを詳細に表示する

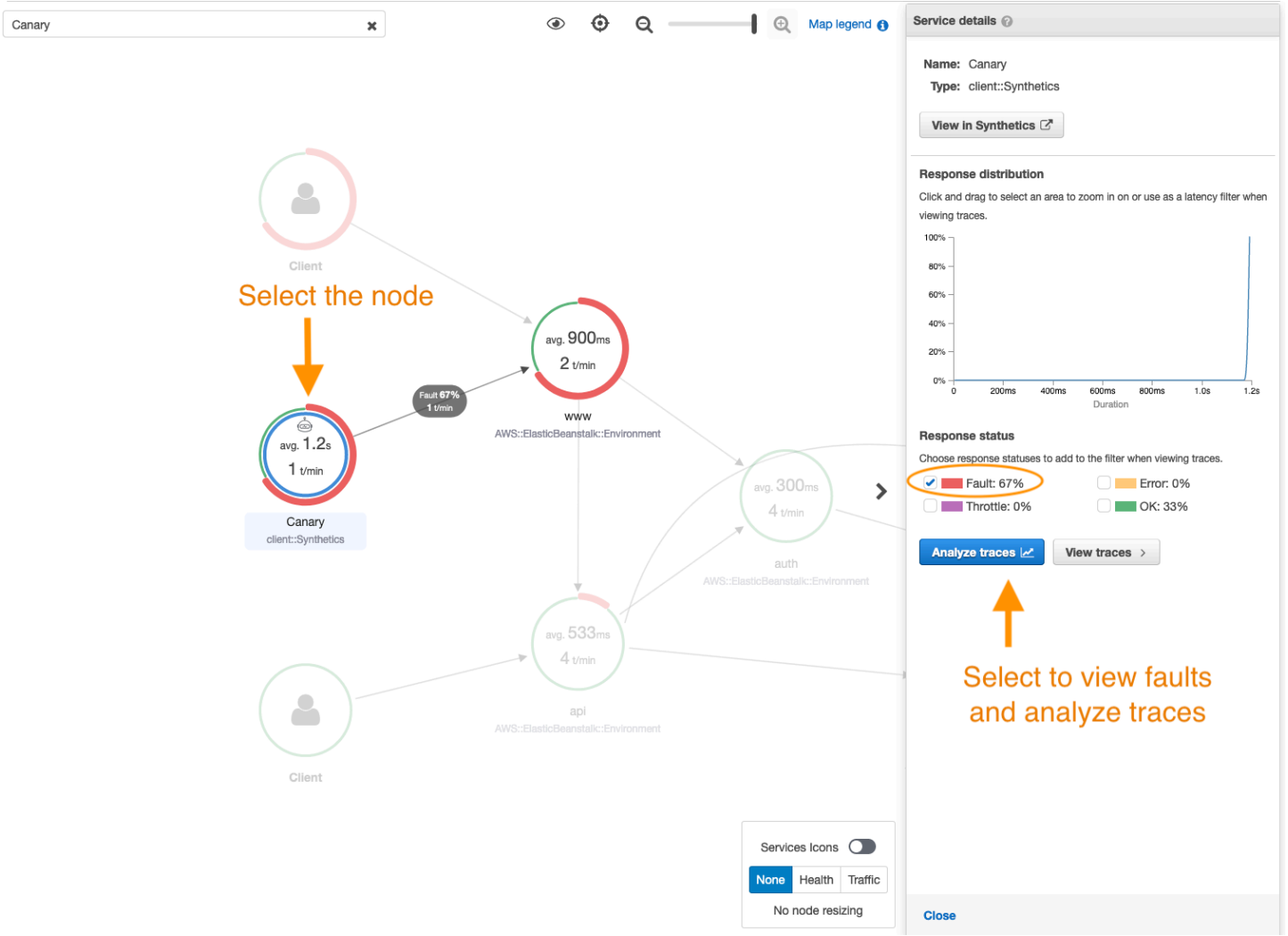
レイテンシーが最も大きいサービスやエラーの原因となっているサービスを特定するには、トレースマップでトレースを選択してトレースの詳細マップを呼び出します。個々のトレース詳細マップには、1 つのリクエストの end-to-end パスが表示されます。このパスを使用して、起動するサービスを把握し、アップストリームおよびダウンストリームサービスを可視化します。

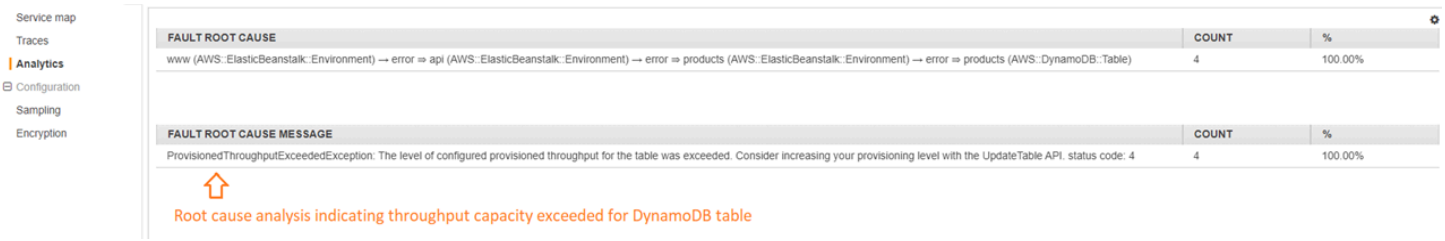


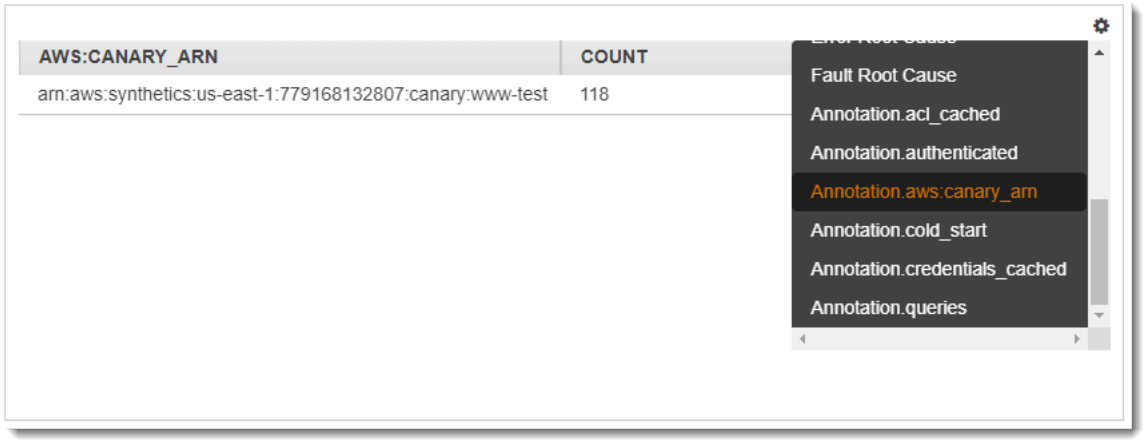
アップストリームおよびダウンストリームサービスで継続的に発生している障害の根本原因を特定する

Synthetics Canary の障害に関する CloudWatch アラームを受け取ったら、X-Ray のトレースデータの統計モデリングを使用して、X-Ray Analytics コンソール内で問題の考えられる根本原因を判断します。Analyticsコンソールno、応答時間の根本原因表には、記録されたエンティティパスを表示します。X-Rayは、トレース内の、どのパスが応答時間の最大の原因であるかを判断します。この形式は、検出されたエンティティの階層を示し、最後に応答時間の根本原因を示します。

以下の例では、API Gateway上で実行されている API 「XXX」のSyntheticsテストが、Amazon DynamoDB表からのスループット容量の例外により障害になっていることを示しています。

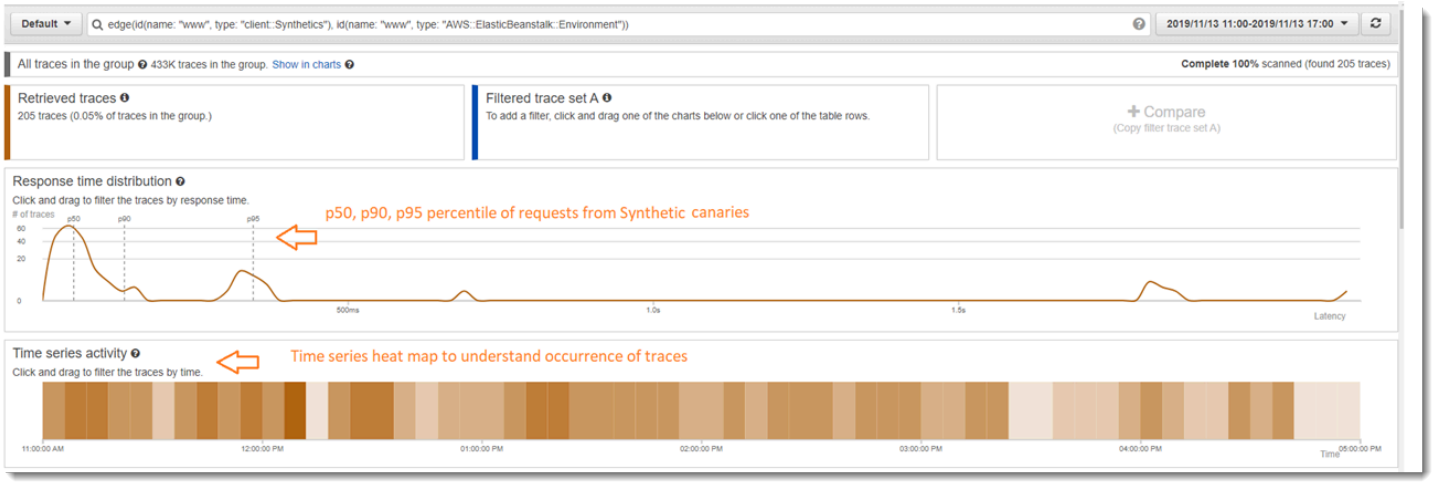

Canary
Map legend
Service details
Name: Canary
Type: client::Synthetics
View in Synthetics
Response distribution
Click and drag to select an area to zoom in on or use as a latency filter when viewing traces.
100%
80%
60%
40%
20%
0%
0
200ms
400ms
600ms
800ms
1.0s
1.2s
Duration
Response status
Choose response statuses to add to the filter when viewing traces.
Fault: 67%
Error: 0%
Throttle: 0%
OK: 33%
Analyze traces
View traces
Select to view faults and analyze traces
Client
Select the node
avg. 900ms
2 t/min
Fault 67%
1 t/min
www
AWS::ElasticBeanstalk::Environment
avg. 1.2s
1 t/min
Canary
client::Synthetics
avg. 300ms
4 t/min
auth
AWS::ElasticBeanstalk::Environment
avg. 533ms
4 t/min
api
AWS::ElasticBeanstalk::Environment
Client
Services Icons
None
Health
Traffic
No node resizing
Close


Service map
Traces
Analytics
Configuration
Sampling
Encryption
FAULT ROOT CAUSE
COUNT
%
www (AWS::ElasticBeanstalk::Environment) ⟶ error ⇒ api (AWS::ElasticBeanstalk::Environment) ⟶ error ⇒ products (AWS::ElasticBeanstalk::Environment) ⟶ error ⇒ products (AWS::DynamoDB::Table)
4
100.00%
FAULT ROOT CAUSE MESSAGE
COUNT
%
ProvisionedThroughputExceededException: The level of configured provisioned throughput for the table was exceeded. Consider increasing your provisioning level with the UpdateTable API. status code: 4
4
100.00%
Root cause analysis indicating throughput capacity exceeded for DynamoDB table



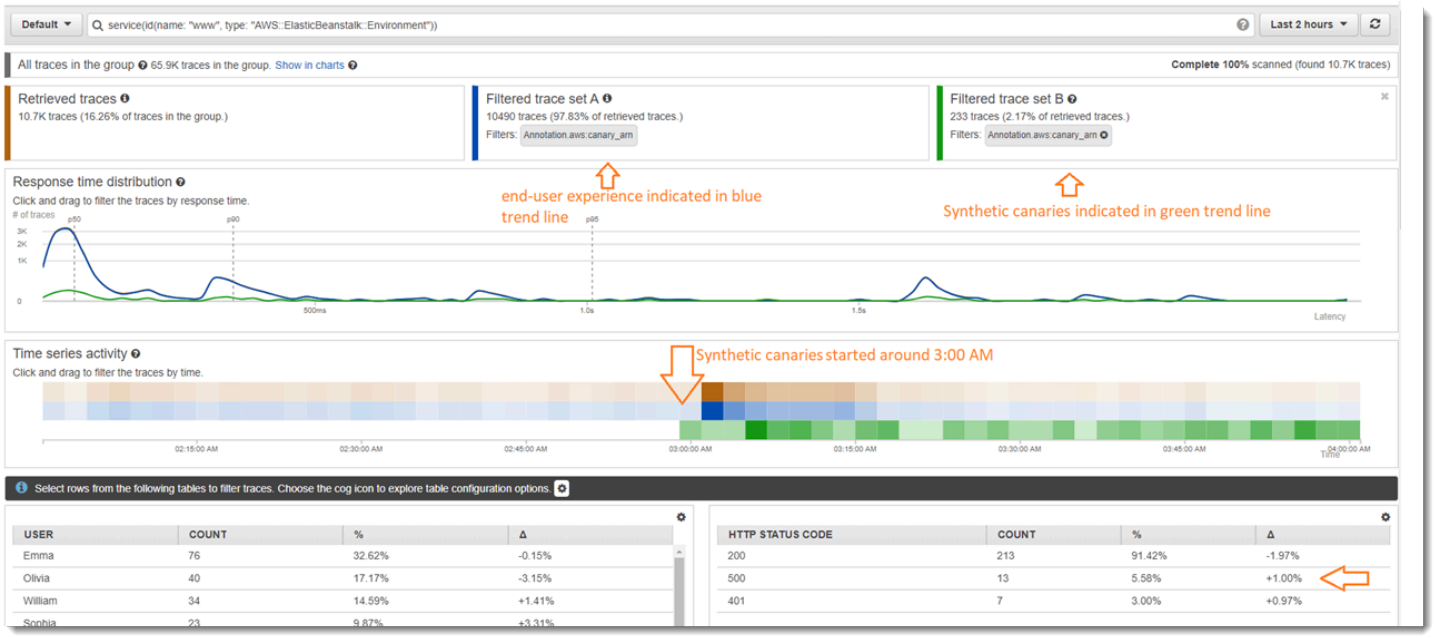
パフォーマンスのボトルネックとトレンドを特定する

Synthetics Canary からの継続的なトラフィックを使用して、一定期間にわたるトレース詳細マップにデータを入力すると、エンドポイントのパフォーマンスの傾向を経時的に表示できます。



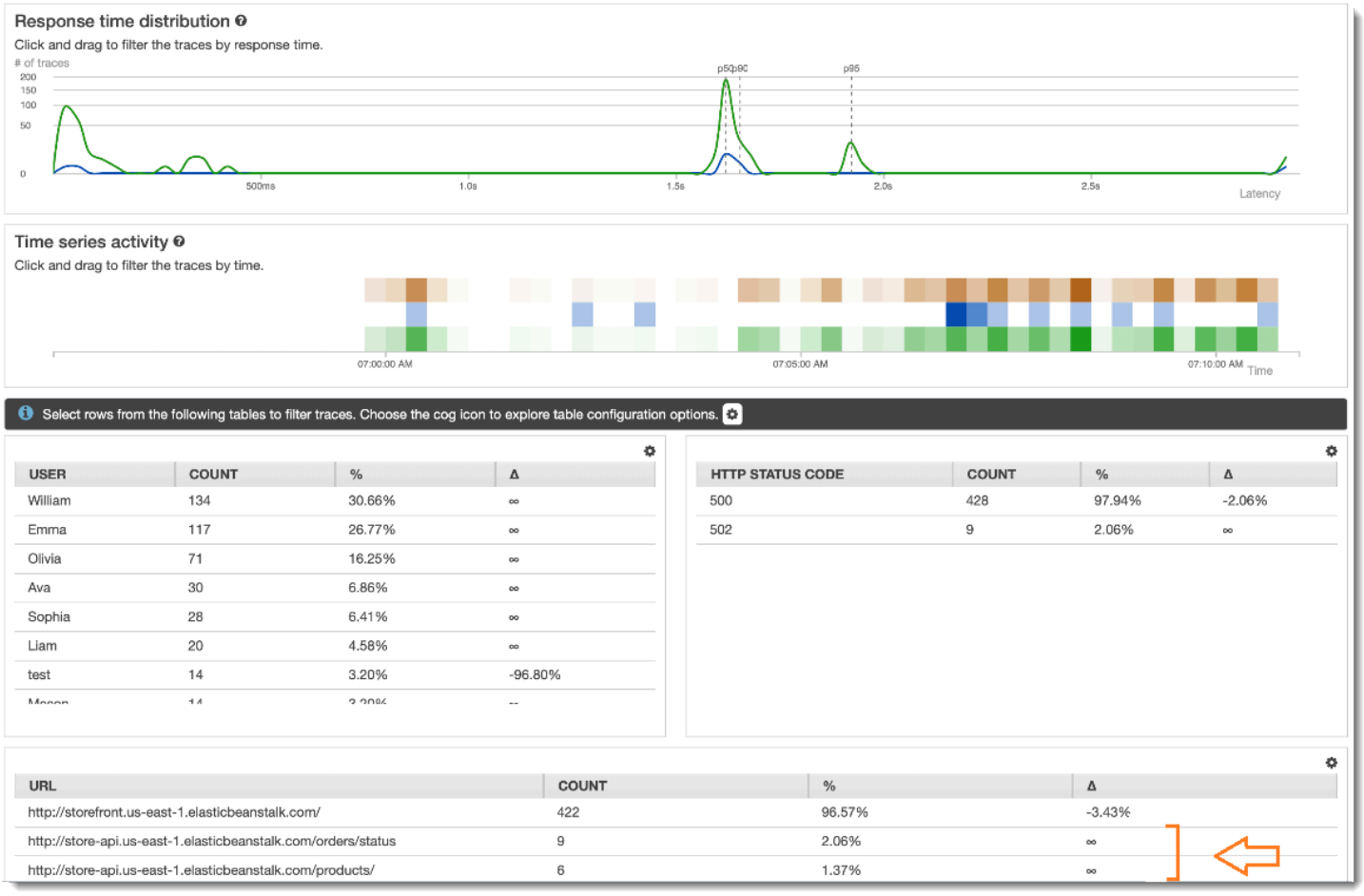
変更前と変更後で待ち時間およびエラー・障害率を比較する

Canary が検出した問題の増加と関連付けるために、変更が発生した時刻を特定します。X-Ray Analyticsコンソールを使用して、前後の時間範囲を異なるトレースセットとして定義し、応答時間分布に視覚的な差異が生じます。



すべてのAPIとURLに必要なcanaryの受信可能範囲エリアを特定する

X-Ray Analyticsを使用して、ユーザー間でカナリアの満足体験を比較します。以下のUIは、カナリアが青いトレンドラインおよびユーザーが緑のトレンドラインを示しています。また、3つのURLのうち2つにcanaryテストがないことを確認できます。

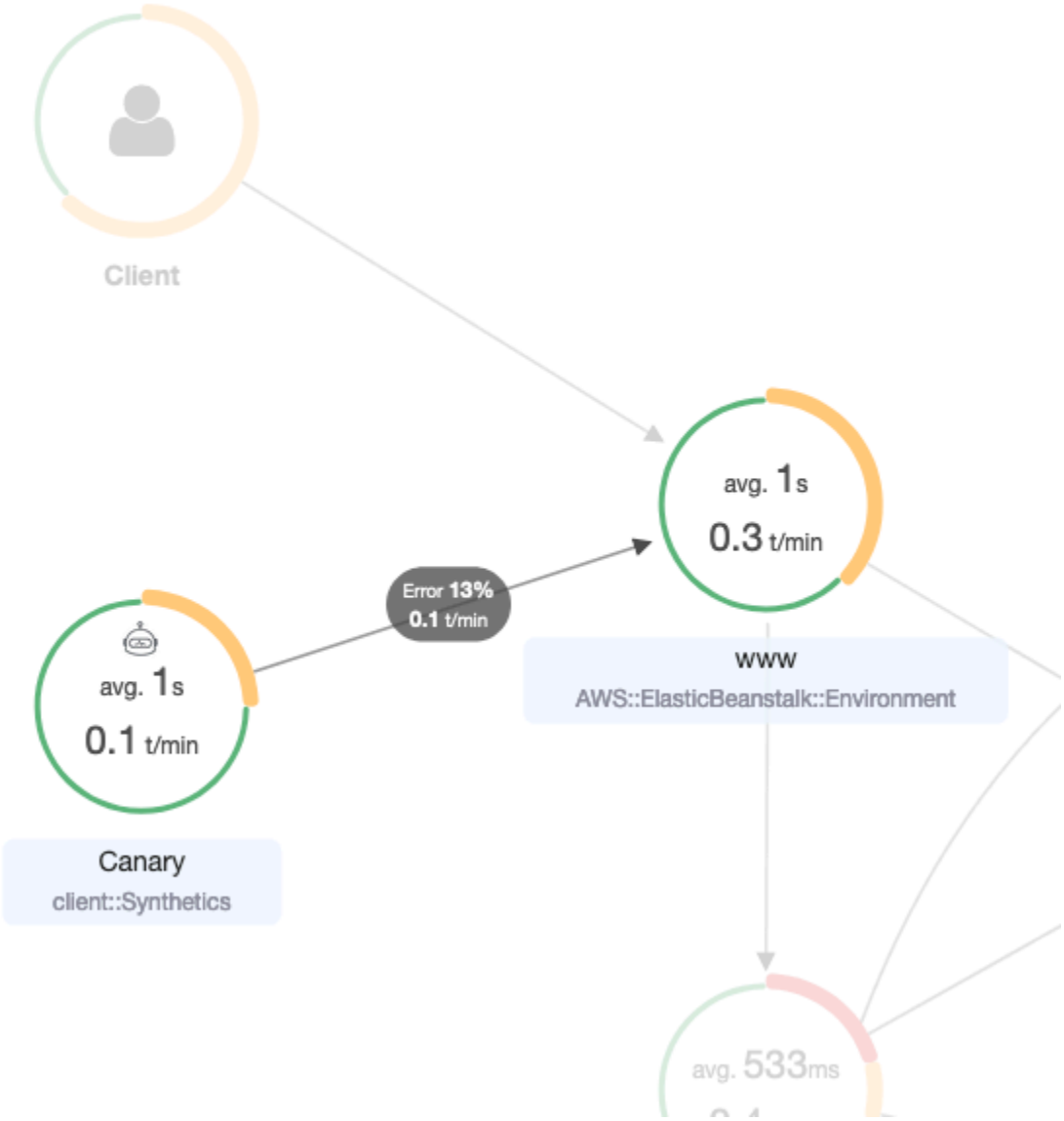


グループを使用してSyntheticsテストに焦点を合わせる

フィルター式を使用してX-Rayグループを作成し、特定のワークフローセットに焦点を合わせることができ、 AWS Elastic Beanstalk例えばNETで実行されているアプリケーション「www」のSyntheticsテストなどです。複雑なキーワード `service()`と を使用して`edge()`、サービスとエッジをフィルタリングします。詳細については、「」の「複雑なキーワード」セクションを参照してください[フィルター式を使用する]。

Example グループフィルタ式

```
"edge(id(name: "www", type: "client::Synthetics"), id(name: "www", type:
 "AWS::ElasticBeanstalk::Environment"))"
```



X-Rayの暗号化設定の変更を追跡AWS Config

AWS X-Ray統合することで、AWS ConfigX-Ray暗号化リソースに対して行われた設定変更を記録することができます。X-Rayの暗号化リソースのインベントリ、AWS ConfigX-Rayの設定履歴の監査、およびリソースの変更に基づく通知の送信などに使用できます。

AWS Configは、以下のX-Ray暗号化リソースの変更をイベントとして記録することをサポートしています：

- 設定変更 暗号化キーの変更および追加、もしくは暗号化X-Ray設定をデフォルトに戻すことができます。

以下の手順に従って、X-Rayとの間で基本的な接続方法をご確認ください。AWS Config

Lambda関数のトリガーの作成

カスタムAWS Lambdaルールを生成する前に、カスタムAWS Config関数のARNを取得する必要があります。以下の手順に従い、AWS Configリソースの状態に基づいて、準拠している値または準拠しない値を`XrayEncryptionConfig`返す基本的な関数をNode.jsで作成します。

AWS::XrayEncryptionConfigの変更トリガーを持つLambda関数を作成するには

1. Lambdaのコンソールを開きます。[機能の作成]を選択します。
2. [設計図]を選択し、設計図ライブラリをフィルタリングして、トリガー変更のルール設定で設計図を探します。設計図名のリンクをクリックするか、[設定]を選択して続行してください。
3. 以下のフィールドを定義して設計図を設定します：
   - [Name] には、名前を入力します。
   - [役割]では、[テンプレートから新しい役割の作成]を選択します。
   - [役割名]に名前を入力します。
   - [ポリシーテンプレート]では、[AWS Configルールのアクセス権限]を選択します。
4. [関数の作成]を選択すると、AWS Lambda 関数が作成され、コンソールに表示されます。
5. 関数コードを編集して、`AWS::EC2::InstanceAWS::XrayEncryptionConfig`に置き換えます。また、説明欄を更新して、この変更を反映させることもできます。

   初期設定コード

```
    if (configurationItem.resourceType !== 'AWS::EC2::Instance') {
        return 'NOT_APPLICABLE';
    } else if (ruleParameters.desiredInstanceType ===
configurationItem.configuration.instanceType) {
        return 'COMPLIANT';
    }
        return 'NON_COMPLIANT';
```

   更新されたコード

```
    if (configurationItem.resourceType !== 'AWS::XRay::EncryptionConfig') {
        return 'NOT_APPLICABLE';
    } else if (ruleParameters.desiredInstanceType ===
configurationItem.configuration.instanceType) {
```

```
        return 'COMPLIANT';
    }
        return 'NON_COMPLIANT';
```

6. X-Rayにアクセスするために、IAMの実行の役割に以下を追加します。これらのアクセス権限により、X-Rayリソースへの読み取り専用のアクセスを許可します。適切なリソースへのアクセス権を提供しない場合、ルールに関連付けられたLambda関数を評価する際に、AWS Config範囲外のメッセージが表示されます。

```
{
    "Sid": "Stmt1529350291539",
    "Action": [
        "xray:GetEncryptionConfig"
    ],
    "Effect": "Allow",
    "Resource": "*"
 }
```

X-Rayに関するカスタムAWS Configルールの作成

Lambda関数が作成されたら、関数のARNをメモし、AWS Configコンソールに移動してカスタムルールを作成します。

X-RayのルーAWS Configルを作成するには

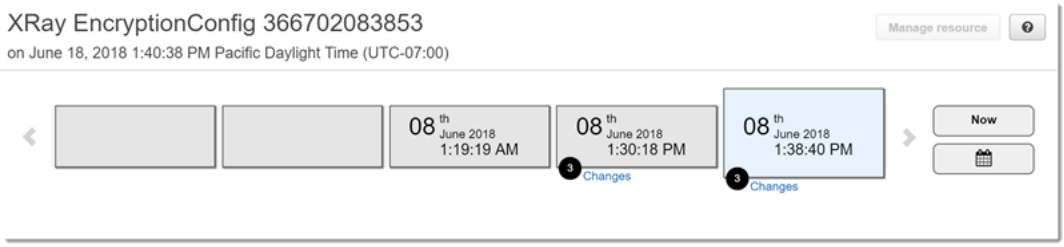
1. コンソールの[ルールAWS Config]ページを開きます。
2. [ルールの追加]を選択し、[カスタムルールを追加]を選択します。
3. AWS Lambda[関数のARN]には、使用するLambda関数に関連付けられた ARNを挿入します。
4. 設定するトリガーの種類を選択します：
   - 設定変更-ルーAWS Configルの範囲に該当するリソースの設定が変更された場合にのトリガーの評価を開始します。AWS Config設定項目の変更通知を送信後に、評価が実行されます。
   - 定期的-指定した間隔(24時間ごとなど)で、AWS Configルールの評価が実行されます。
5. [リソースタイプ]は、EncryptionConfigX-Rayの項目で選択します。
6. [セーブ]を選択します。

AWS Config コンソールは、ルールのコンプライアンスの評価を直ちに開始します。評価は完了までに数分かかることがあります。

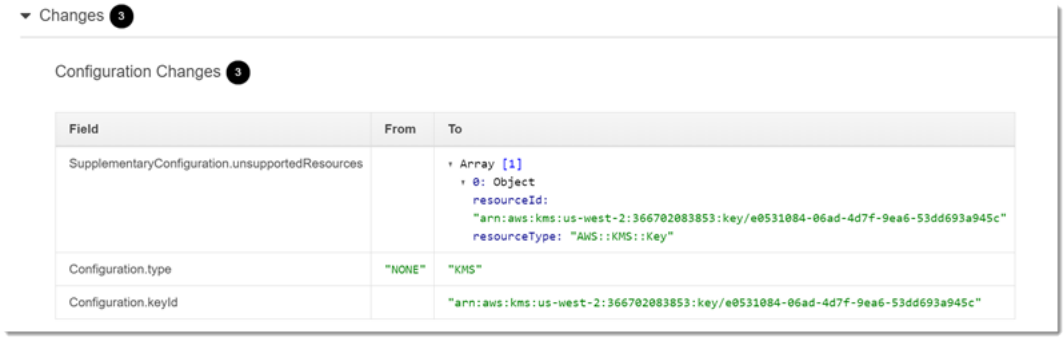
このルールに準拠し、AWS Config監査履歴の作成を開始できるようになります。AWS Configは、タイムラインの形式でリソースの変更を記録します。AWS Config はイベントのタイムラインの各変更に対して、変更前/変更後の形式で表を作成し、暗号化キーの JSON 表現の変更内容を示します。EncryptionConfigに関連付けられた2つのフィールドの変更は`Configuration.type`及び`Configuration.keyID`。

結果の例

以下は、特定の日時に行われた変更を示すAWS Configタイムラインの例です。



以下は、AWS Config変更エントリの例です。変更前/変更後の形式で変更内容を示します。この例では、デフォルトのX-Ray暗号化設定を、定義済みの暗号化キーに変更されたことを示しています。



Amazon SNSの通知

設定変更の通知を受け取るには、AWS ConfigAmazon SNS通知を発行するように設定します。詳細については、「EメールによるAWS Configリソースの変更のモニタリング」を参照してください。

Amazon Elastic Compute CloudおよびAWS X-Ray

AmazonEC2インスタンスにX-Rayデーモンをインストールし、ユーザーデータスクリプトを使用して実行することができます。手順については、「Amazon EC2 での X-Ray デーモンの実行」を参照してください。

インスタンスプロファイルを使用して、デーモンにX-Rayのトレースデータをアップロードするアクセス権限を付与します。詳細については、「X-Rayにデータを送信するアクセス権限をデーモンに付与する」を参照してください。

AWS Elastic BeanstalkおよびAWS X-Ray

AWS Elastic Beanstalkプラットフォームには、X-Rayデーモンが含まれます。デーモンは、Elastic Beanstalkのコンソールで、オプションを設定するか、または環境設定ファイルで実行することができます。

Java SEプラットフォームでは、Buildfileファイルを使用して、MavenまたはGradleをオンインスタンスで使用するアプリケーションを構築することができます。X-Ray SDK for JavaAWS SDK for JavaおよびMavenから使用できるため、すべての依存関係を一括してアップロードする必要はなく、アプリケーションコードのみを配置してオンインスタンスで構築することができます。

Elastic Beanstalkの環境プロパティを使用して、X-Ray SDKを設定できます。Elastic Beanstalkが環境プロパティをアプリケーションに渡すために使用する方法は、プラットフォームによって異なります。お使いのプラットフォームに応じて、X-Ray SDK'sの環境変数または、異なるシステムプロパティを使用してください。

- Node.jsプラットフォーム・環境変数を使用する
- Java SEプラットフォーム・環境変数を使用する
- Tomcatプラットフォーム・システムプロパティを使用する

詳細については、「開発者ガイド」AWS X-Rayのデバッグの設定AWS Elastic Beanstalkを参照してください。

Elastic Load Balancing と AWS X-Ray

Elastic Load Balancingアプリケーションロードバランサーは、受信するHTTPリクエストに、ヘッダーにトレースIDを追加します。`X-Amzn-Trace-Id`

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

X-Ray トレース ID 形式

X-Ray `trace_id` は、ハイフンで区切られた 3 つの数字で構成されています。例えば、`1-58406520-a006649127e371903a2de979` と指定します。これには、以下のものが含まれます：

- バージョン番号。 1
- 8 桁の 16 進数 を使用した Unix エポックタイムでの元のリクエストの時刻。

  例えば、2016 年 12 月 1 日午前 10:00 PST のエポックタイムは `1480615200`秒または 16 進数`58406520`です。
- 24 桁の 16 進数のトレース のグローバルに一意の 96 ビット識別子。

ロードバランサーはX-Rayにデータを送信しない場合、サービスマップ上にノードとして表示されません。

詳細については、『Elastic Load Balancing Developer Guide』の「Application Load Balancerに関するリクエストのトレース」を参照してください。

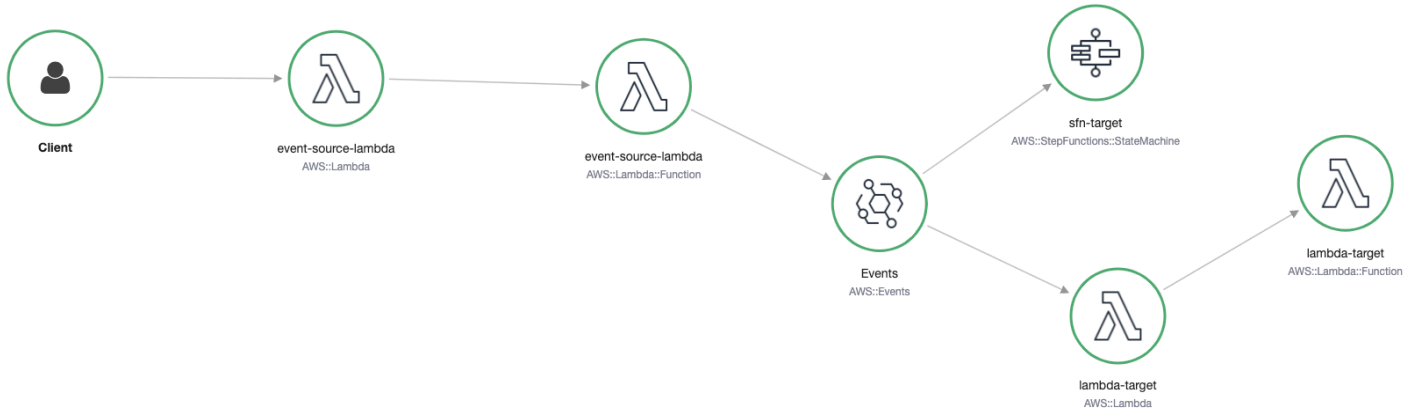
Amazon EventBridge と AWS X-Ray

AWS X-Ray は Amazon と統合して EventBridge 、 を通過するイベントをトレースします EventBridge。X-Ray SDK で計測されたサービスが にイベントを送信する場合 EventBridge、トレースコンテキストはトレースヘッダー 内のダウンストリームイベントターゲットに伝播されます。X-Ray SDKは、自動的にトレースヘッダーを取得し、後続のインストルメンテーションに適用します。この継続性により、ユーザーはダウンストリームサービス全体でトレース、分析、およびデバッグを実行できます。また、システムの全体像を把握できるようになります。

詳細については、「 ユーザーガイド」のEventBridge 「X-Ray 統合EventBridge 」を参照してください。

X-Ray サービスマップでのソースおよびターゲットの表示

X-Ray トレースマップには、ソースサービスとターゲットサービスを接続する EventBridge イベントノードが表示されます。詳細については、「X-Ray トレースマップを使用する」を参照してください。トレースマップの例を次に示します。



トレースコンテキストをイベントターゲットに伝播する

X-Ray SDK を使用すると、 EventBridge イベントソースはトレースコンテキストをダウンストリームのイベントターゲットに伝達できます。次の言語固有の例は、アクティブトレースが有効になっている Lambda 関数 EventBridge から を呼び出す方法を示しています。

Java

X-Rayに必要な依存関係を追加します：

- AWS X-Ray SDK for Java
- AWS X-Ray Recorder SDK for 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.getStackTrace();
```

```
    }
    return "success";
  }
}
```

Python

requirements.txt ファイルに以下の依存関係を追加します：

```
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#

以下のX-Rayパッケージを C# の依存関係に追加します：

```
<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 および AWS X-Ray

AWS X-Ray を使用して、 AWS Lambda 関数を追跡できます。Lambda は X-Ray デーモンを実行し、関数の呼び出しと実行に関する詳細でセグメントを記録します。さらに計測するには、X-Ray SDK を関数にバンドルして送信呼び出しを記録し、注釈とメタデータを追加できます。

Lambda 関数が他の計測サービスより呼び出されると、Lambda は、追加の設定なしで、すでにサンプルを作成したリクエストをトレースします。アップストリームサービスは、計測されたウェブアプリケーションや別の Lambda 関数とすることができます。サービスは、計測された AWS SDK クライアントを使用して関数を直接呼び出すか、計測された HTTP クライアントを使用して API Gateway API を呼び出すことによって呼び出すことができます。

AWS X-Ray は、 AWS Lambda および Amazon SQS を使用したイベント駆動型アプリケーションのトレースをサポートします。 CloudWatch コンソールを使用して、Amazon SQS でキューに入れられ、ダウンストリーム Lambda 関数によって処理される各リクエストの接続されたビューを表示します。アップストリームメッセージプロデューサーからのトレースは、ダウンストリーム Lambda コンシューマーノードからのトレースに自動的にリンクされ、アプリケーションの end-to-end ビューが作成されます。詳細については、「イベント駆動型アプリケーションのトレース」を参照してください。

> **Note**
>
> ダウンストリーム Lambda 関数でトレースを有効にしている場合は、ダウンストリーム関数がトレースを生成するために、ダウンストリーム関数を呼び出すルート Lambda 関数でもトレースを有効にする必要があります。

Lambda 関数がスケジュールで実行される場合や、計測されていないサービスによって呼び出される場合は、アクティブトレースで呼び出しをサンプルおよび記録するように Lambda を設定できます。

AWS Lambda 関数で X-Ray 統合を設定するには

1. AWS Lambda コンソールを開きます。
2. 左のナビゲーションペインから [関数] を選択します。
3. 関数を選択します。
4. 設定 タブで、追加のモニタリングツールカードまでスクロールダウンします。このカードは、左側のナビゲーションペインでモニタリングツールとオペレーションツールを選択して検索することもできます。
5. [Edit] (編集) を選択します。
6. [AWS X-Ray] で、[アクティブトレース] を有効にします。

対応する X-Ray SDK、 Lambda があるランタイムでも、X-Ray デーモンが実行されます。

Lambda の X-Ray SDK

- Go X-Ray SDK for Go – Go 1.7 以降のランタイム
- X-Ray SDK for Java – Java 8 ランタイム
- X-Ray SDK for Node.js – Node.js 4.3 以降のランタイム
- X-Ray SDK for Python – Python 2.7、Python 3.6 以降のランタイム
- X-Ray SDK for .NET – .NET Core 2.0 以降のランタイム

Lambda で X-Ray SDK を使用するには、新しいバージョンを作成するたびに、関数コードでバンドルします。Lambda 関数は、他のサービスで実行されているアプリケーションを計測する場合と同じメソッドを使用して計測することができます。主な違いは、SDK を使用して、受信リクエストの計測、サンプリングの決定、セグメントの作成を行わないことです。

Lambda 関数とウェブアプリケーションの計測のもう 1 つの違いは、Lambda が作成して X-Ray に送信するセグメントは、関数コードで変更できないことです。サブセグメントを作成し、そこに注釈とメタデータを記録できますが、親セグメントに注釈とメタデータを追加することはできません。

詳細については、AWS 開発者ガイドの「AWS Lambda X-Ray を使用する」を参照してください。

Amazon SNS と AWS X-Ray

Amazon Simple Notification Service (Amazon SNS) AWS X-Ray で を使用すると、SNS トピックから SNS がサポートするサブスクリプションサービスに移動するリクエストを追跡および分析できます。 Amazon SNS Amazon SNS と X-Ray トレースを併用して、リクエストがトピックに費やされる時間や、トピックの各サブスクリプションにメッセージを配信するのにかかった時間など、メッセージとそのバックエンドサービスのレイテンシーを分析できます。Amazon SNS は、標準トピックと FIFO トピックの両方で X-Ray トレースをサポートしています。

X-Ray で既に計測されているサービスから Amazon SNS トピックに発行すると、Amazon SNS はトレースコンテキストをパブリッシャーからサブスクライバーに渡します。さらに、アクティブトレースを有効にして、計測された SNS クライアントから発行されたメッセージの Amazon SNS サブスクリプションに関するセグメントデータを X-Ray に送信できます。Amazon SNS コンソール、または Amazon SNS API か CLI を使用して、Amazon SNS トピックのアクティブトレースを有効にします。SNS クライアントの計測の詳細については、「アプリケーションの計測」を参照してください。

Amazon SNS アクティブトレースの設定

Amazon SNS コンソール、 AWS CLI または SDK を使用して、Amazon SNS アクティブトレースを設定できます。

Amazon SNS コンソールを使用する場合、Amazon SNS は SNS が X-Ray を呼び出すために必要なアクセス許可の作成を試みます。X-Ray リソースポリシーを変更するための十分なアクセス許可がない場合、この試行は拒否されることがあります。これらのアクセス許可の詳細については、「Amazon Simple Notification Service デベロッパーガイド」の「Amazon SNS での Identity and Access Management」および「Amazon SNS アクセスコントロールのケース例」を参照してください。Amazon SNS コンソールを使用してアクティブトレースを有効にする方法の詳細については、「Amazon Simple Notification Service デベロッパーガイド」の「Amazon SNS トピックでアクティブトレースを有効にする」を参照してください。

AWS CLI または SDK を使用してアクティブトレースを有効にする場合は、リソースベースのポリシーを使用してアクセス許可を手動で設定する必要があります。`PutResourcePolicy` を使用し

て、Amazon SNS が X-Ray にトレースを送信できるようにするために必要なリソースベースのポリシーを使用して X-Ray を設定します。

Example Amazon SNS アクティブトレース用の X-Ray リソースベースのポリシーの例

このポリシードキュメントの例では、Amazon SNS がトレースデータを X-Ray に送信するために必要なアクセス許可を指定します。

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

CLI を使用して、Amazon SNS アクセス許可を付与するリソースベースのポリシーを作成して、トレースデータを X-Ray に送信します。

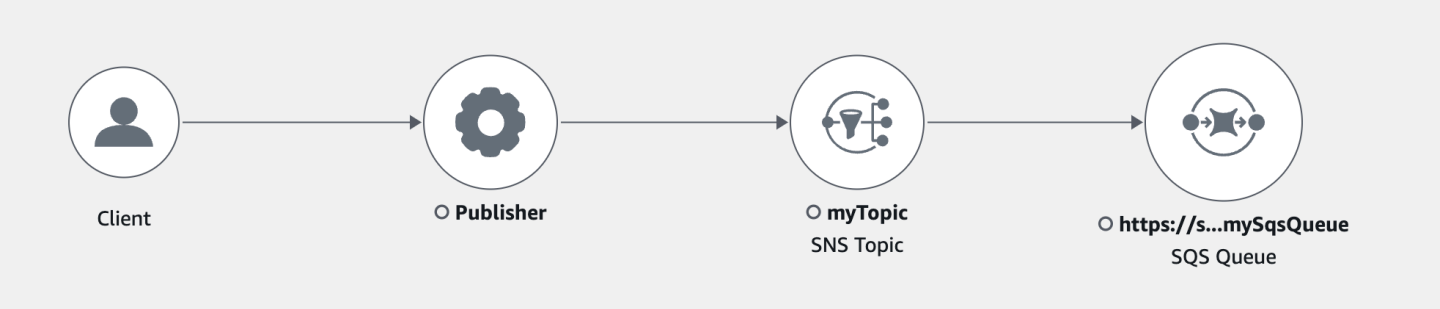
```
aws xray put-resource-policy --policy-name MyResourcePolicy --policy-document
 '{ "Version": "2012-10-17", "Statement": [ { "Sid": "SNSAccess", "Effect": "Allow",
 "Principal": { "Service": "sns.amazonaws.com" }, "Action": [ "xray:PutTraceSegments",
 "xray:GetSamplingRules", "xray:GetSamplingTargets" ], "Resource": "*",
```

```
"Condition": { "StringEquals": { "aws:SourceAccount": "account-id" }, "StringLike":
{ "aws:SourceArn": "arn:partition:sns:region:account-id:topic-name" } } } ] }'
```

これらの例を使用するには、*partition*、*region*、*account-id*、 *topic-name*を特定の AWS パーティション、リージョン、アカウント ID、および Amazon SNS トピック名に置き換えます。すべての Amazon SNS トピックに、トレースデータを X-Ray に送信するアクセス許可を付与するには、トピック名を * に置き換えます。

X-Ray コンソールで Amazon SNS パブリッシャートレースとサブスクライバーのトレースを表示する

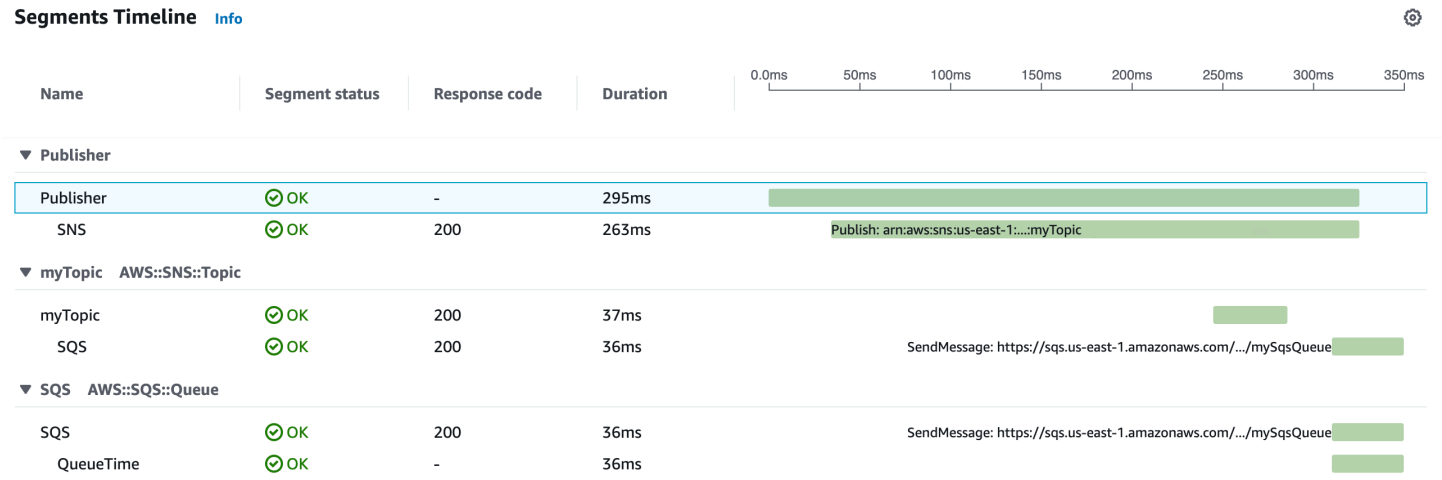
X-Ray コンソールを使用して、Amazon SNS パブリッシャーとサブスクライバーの接続されたビューを表示するトレースマップとトレースの詳細を表示します。トピックに対して Amazon SNS アクティブトレースが有効になっている場合、X-Ray トレースマップとトレース詳細マップには、Amazon SNS パブリッシャー、Amazon SNS トピック、ダウンストリームサブスクライバーの接続されたノードが表示されます。



Amazon SNS パブリッシャーとサブスクライバーにまたがるトレースを選択すると、X-Ray トレースの詳細ページにトレースの詳細マップとセグメントタイムラインが表示されます。

Example Amazon SNS パブリッシャーとサブスクライバーのタイムラインの例

この例は、Amazon SNS トピックにメッセージを送信する Amazon SNS パブリッシャーを含むタイムラインを示しています。このメッセージは Amazon SQS サブスクライバーによって処理されます。

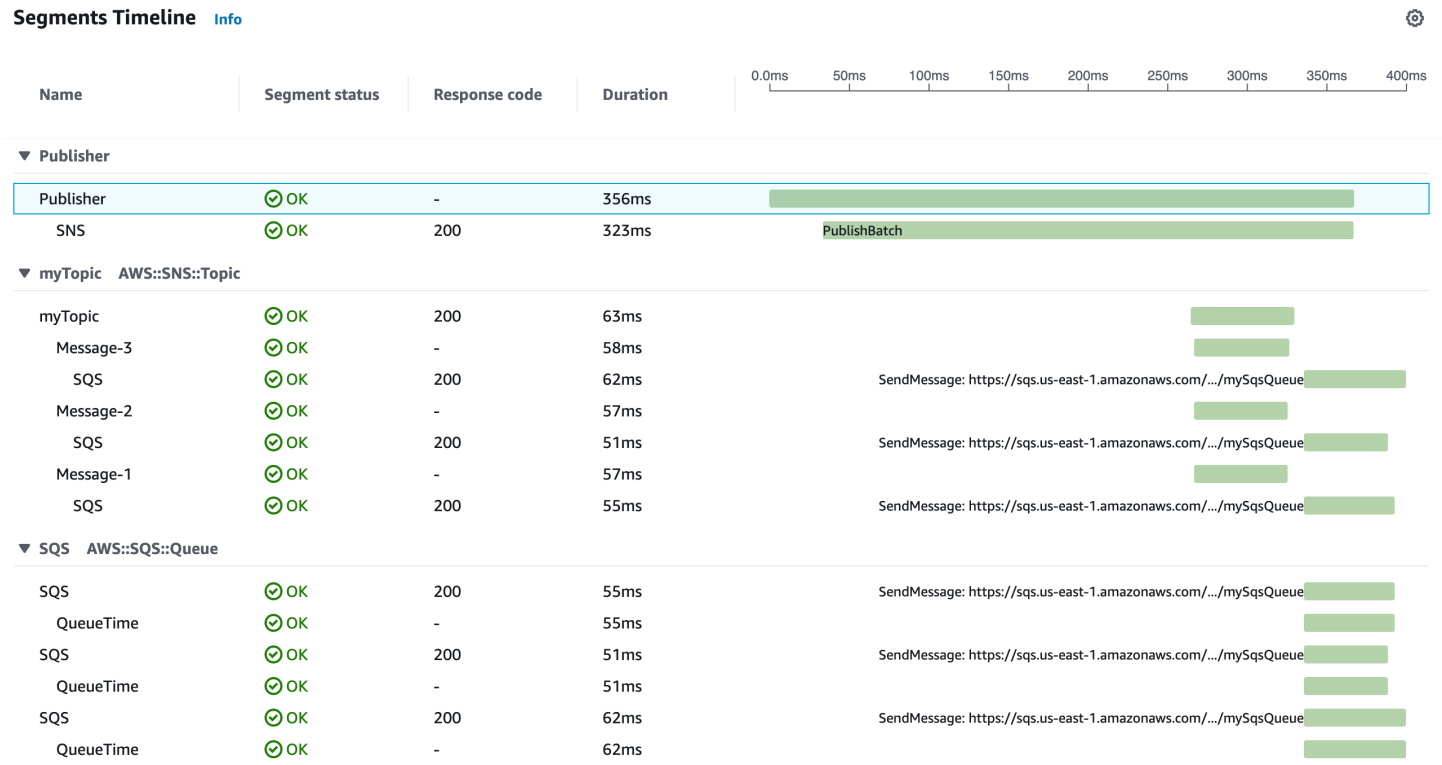


上記のタイムラインの例は、Amazon SNS メッセージフローの詳細を示しています。

- SNS セグメントは、クライアントからの `Publish` API 呼び出しの往復所要時間を表します。
- myTopic セグメントは、発行リクエストに対する Amazon SNS レスポンスのレイテンシーを表します。
- SQS サブセグメントは、Amazon SNS が Amazon SQS キューにメッセージを発行するのにかかる往復時間を表します。
- myTopic セグメントと SQS サブセグメントの間隔は、メッセージが Amazon SNS システムで費やす時間を表します。

Example Amazon SNS メッセージのバッチ処理を含むタイムラインの例

複数の Amazon SNS メッセージが 1 つのトレース内でバッチ処理される場合、セグメントタイムラインには、処理された各メッセージを表すセグメントが表示されます。



AWS Step Functions および AWS X-Ray

AWS X-Ray は AWS Step Functions と統合して、Step Functions のリクエストをトレースおよび分析します。ステートマシーンのコンポーネントの視覚化、パフォーマンスのボトルネックの特定、およびエラーの原因となったリクエストのトラブルシューティングを行うことができます。詳細については、デベロッパーガイドの AWS X-Ray と Step Functions AWS Step Functionsを参照してください。

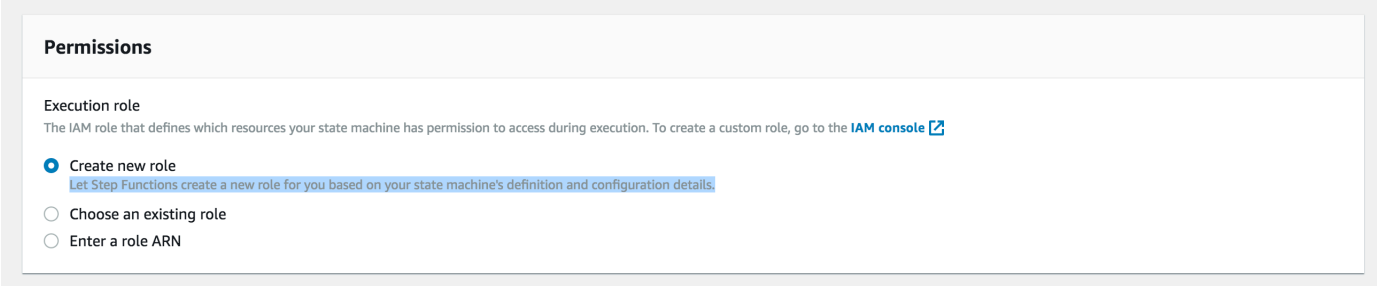
新しいステートマシンの作成時に X-Ray トレースを有効にするには

1. https://console.aws.amazon.com/states/ で Step Functions コンソールを開きます。
2. [ステートマシンの作成] を選択します。
3. [ステートマシンを定義する] ページで、[コードスニペットで作成] および [テンプレートで開始する] を選択します。サンプルプロジェクトを実行することを選択した場合、作成中に X-Ray トレースを有効にすることはできません。の代わりに、ステートマシンを作成した後で X-Ray トレースを有効にします。
4. [Next] (次へ) をクリックします。
5. [詳細の指定] ページで、ステートマシンを設定します。

6. [X-Ray トレースを有効にする] を選択します。

既存のステートマシンで、X-Ray トレースを有効にするには

1. Step Functions コンソールで、トレースを有効にするステートマシンを選択します。
2. [Edit] (編集) を選択します。
3. [X-Ray トレースを有効にする] を選択します。
4. （オプション）[Permissions] ウィンドウから [Create new role] を選択して、X-Ray のアクセス許可を含むステートマシンの新しいロールを自動生成します。



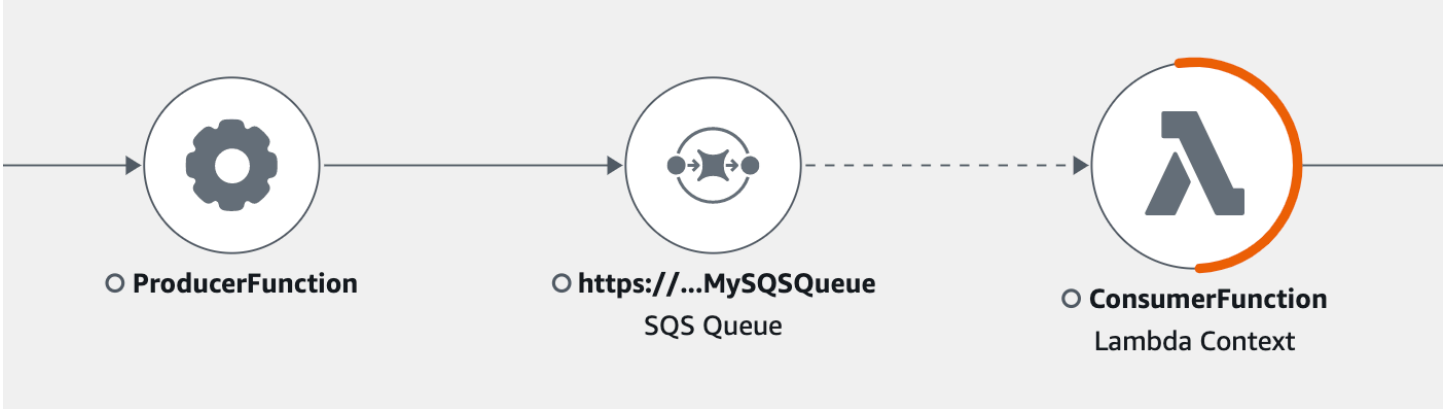
5. [Save (保存)] を選択します。

> **Note**
>
> 新しいステートマシンを作成すると、リクエストがサンプリングされ、Amazon API Gateway または AWS Lambda などのアップストリームサービスでトレースが有効になっていれば、自動的にトレースされます。AWS CloudFormation テンプレートなど、コンソール以外で設定された既存のステートマシンの場合、X-Ray トレースを有効にするための十分なアクセス許可を付与する IAM ポリシーがあることを確認します。

Amazon SQS と AWS X-Ray

AWS X-Ray は Amazon Simple Queue Service (Amazon SQS) と統合して、Amazon SQS キューを通過するメッセージをトレースします。サービスが X-Ray SDK を使用してリクエストをトレースする場合、Amazon SQS はトレースヘッダーを送信し、整合性のあるトレース ID を持つコンシューマーに、送信者から元のトレースを伝達し続けます。トレースの継続性により、ユーザーはダウンストリームサービス全体でトレース、分析、およびデバッグを実行できます。

AWS X-Ray は、Amazon SQS および を使用したイベント駆動型アプリケーションのトレースをサポートしています AWS Lambda。 CloudWatch コンソールを使用して、Amazon SQS でキューに入れられ、ダウンストリーム Lambda 関数によって処理される各リクエストの接続されたビューを表示します。アップストリームメッセージプロデューサーからのトレースは、ダウンストリーム Lambda コンシューマーノードからのトレースに自動的にリンクされ、アプリケーションの end-to-end ビューが作成されます。詳細については、「イベント駆動型アプリケーションのトレース」を参照してください。



Amazon SQS では、次のトレースヘッダー計測がサポートされています。

- デフォルトの HTTP ヘッダー – X-Ray SDK は、 AWS SDK を通じて Amazon SQS を呼び出すと、トレースヘッダーを HTTP ヘッダーとして自動的に入力します。デフォルトのトレースヘッダーは `X-Amzn-Trace-Id` によって転送され、`SendMessage` または `SendMessageBatch` リクエストに含まれるすべてのメッセージに対応します。デフォルトの HTTP ヘッダーの詳細については、「トレースヘッダー」を参照してください。
- **`AWSTraceHeader`** システム属性 — `AWSTraceHeader` は、Amazon SQS によって予約されたメッセージシステム属性で、X-Ray トレースヘッダーをキューのメッセージとともに渡します。`AWSTraceHeader` は、新しい言語のトレース SDK を構築する場合など、X-Ray SDK による自動計測ができない場合でも使用できます。両方のヘッダー計測が設定されている場合、メッセージシステム属性が HTTP トレースヘッダーを上書きします。

Amazon EC2 で実行した場合、Amazon SQS は一度に 1 つのメッセージの処理をサポートします。これは、オンプレミスホストで実行されている場合、および 、Amazon ECS AWS Fargate、 などのコンテナサービスを使用する場合に適用されます AWS App Mesh。

トレースヘッダーは、Amazon SQS メッセージサイズとメッセージ属性のクォータの両方から除外されます。X-Ray トレースを有効にしても、Amazon SQS クォータを超えることはありません。AWS クォータの詳細については、「Amazon SQS クォータ」を参照してください。

HTTP トレースヘッダーの送信

Amazon SQS の送信者コンポーネントは、`SendMessageBatch` または `SendMessage` 呼び出しを通じて、自動的にトレースヘッダーを送信できます。 AWS SDK クライアントを計測すると、X-Ray SDK でサポートされているすべての言語で自動的に追跡できます。これらのサービス (Amazon S3 バケットや Amazon SQS キューなど) 内でアクセスするトレースされた AWS のサービス および リソースは、X-Ray コンソールのトレースマップにダウンストリームノードとして表示されます。

任意の言語で AWS SDK 呼び出しをトレースする方法については、サポートされている SDKsの以下のトピックを参照してください。

- Go – X-Ray AWS SDK for Go を使用した SDK 呼び出しのトレース
- Java – X-Ray AWS SDK for Java を使用した SDK 呼び出しのトレース
- Node.js – X-Ray AWS SDK for Node.js を使用した SDK 呼び出しのトレース
- Python – X-Ray AWS SDK for Python を使用した SDK 呼び出しのトレース
- Ruby – X-Ray AWS SDK for Ruby を使用した SDK 呼び出しのトレース
- .NET – X-Ray AWS SDK for .NET を使用した SDK 呼び出しのトレース

トレースヘッダーを取得し、トレースコンテキストを復元する

Lambda ダウンストリームコンシューマーを使用している場合、トレースコンテキストの伝播は自動的に行われます。他の Amazon SQS コンシューマーとのコンテキスト伝達を続行するには、レシーバーコンポーネントへの引き継ぎを手動で計測する必要があります。

トレースコンテキストを復元するには、主に 3 つのステップがあります。

- `AWSTraceHeader` APIを呼び出して `ReceiveMessage` 属性のキューからメッセージを受信します。
- 属性からトレースヘッダーを取得します。
- ヘッダーからトレース ID を復元します。必要に応じて、セグメントにメトリクスを追加します。

以下は、X-Ray SDK for Java で記述された実装の例です。

Example : トレースヘッダーを取得し、トレースコンテキストを復元する

```
// 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 と AWS X-Ray

AWS X-Ray は Amazon S3 と統合してアップストリームリクエストをトレースし、アプリケーションの S3 バケットを更新します。サービスが X-Ray SDK を使用してリクエストをトレースする場合、Amazon S3 は AWS Lambda、Amazon SQS、Amazon SNS などのダウンストリームイベントサブスクライバーにトレースヘッダーを送信できます。X-Ray は Amazon S3 イベント通知のトレースメッセージを有効にします。

X-Ray トレースマップを使用して、Amazon S3 とアプリケーションが使用する他のサービス間の接続を表示できます。コンソールを使用して、平均レイテンシーや障害発生率などのメトリクスを表示することもできます。X-Ray コンソールの詳細については、「X-Ray コンソールの詳細」を参照してください。

Amazon S3 は、デフォルトの HTTP ヘッダーの計測をサポートしています。X-Ray SDK は、 AWS SDK を介して Amazon S3 を呼び出すと、トレースヘッダーを HTTP ヘッダーとして自動的に入力します。デフォルトのトレースヘッダーは、`X-Amzn-Trace-Id` によって伝送されます。トレースヘッダーの詳細については、コンセプトページの「トレースヘッダー」を参照してください。Amazon S3 トレースコンテキストの伝播では、Lambda、SQS、および SNS のサブスクライ

バーがサポートされます。SQS と SNS はセグメントデータ自体を出力しないため、トレースヘッダーをダウンストリームサービスに伝達する場合でも、S3 によってトリガーされるとトレースマップやトレースマップには表示されません。

Amazon S3 イベント通知を設定する

Amazon S3 通知機能で、バケット内で の特定のイベントが発生したときに、通知を受けることができます。これらの通知は、アプリケーション内の次の宛先に伝播できます。

- Amazon Simple Notification Service (Amazon SNS)
- Amazon Simple Queue Service (Amazon SQS)
- AWS Lambda

サポートされているイベントのリストについては、Amazon S3 開発者ガイドでサポートされているイベントタイプを参照してください。。

Amazon SNS と Amazon SQS

SNS トピックや SQS キューに通知を発行するには、まず Amazon S3 のアクセス許可を付与する必要があります。これらのアクセス許可を付与するには、 AWS Identity and Access Management (IAM) ポリシーを送信先の SNS トピックまたは SQS キューにアタッチします。必要な IAM ポリシーの詳細については、SNS トピックまたは SQS キューにメッセージを発行するアクセス許可の付与を参照してください。

SNS および SQS の X-Ray との統合については、Amazon SNS と AWS X-Ray および Amazon SQS と AWS X-Ray を参照してください。

AWS Lambda

Amazon S3 コンソールを使用して、Lambda 関数で S3 バケットのイベント通知を設定する場合、コンソールは Lambda 関数で必要なアクセス許可を設定し、Amazon S3 がバケットから関数を呼び出すアクセス許可を持つようにします。詳細については、Amazon Simple Storage Service コンソールユーザーガイドの「S3 バケットのイベント通知を有効化および設定する方法」を参照してください。

から Amazon S3 に Lambda 関数 AWS Lambda を呼び出すアクセス許可を付与することもできます。詳細については、AWS 「Lambda デベロッパーガイド」の「チュートリアル: Amazon S3 で AWS Lambda を使用する」を参照してください。

Lambda と X-Ray の統合の詳細については、「Lambda での Java AWS コードの計測」を参照してください。

X-Ray でリソースを管理する

このガイドでは、テンプレートを使用して X-Ray のリソースを管理する方法を示します。そのためには、リソースを設定し、キーとオプションの値のペアを使用してリソースにタグ付けします。

AWS CloudFormation テンプレート を使用して、リソースとインフラストラクチャを自動的にセットアップおよび管理できます。このテンプレートを JSONまたは YAML形式で使用して、X-Ray グループ、サンプリングルール、またはリソースポリシーを 1 つのファイルに作成します。例えば、次の AWS CloudFormation テンプレートを使用できます。

- 1 つのテンプレートを使用して、複数のデプロイ間でリソースを一貫して設定し、手動設定エラーを回避します。
- テンプレートファイルを使用して、 AWS アカウント間、開発環境間でリソースを管理し、チーム間でテンプレートファイルを共有します。
- バージョン管理を使用してテンプレートへの変更を制御および追跡し、必要に応じて変更を元に戻します。

タグに基づいてリソースを検索およびフィルタリングし、タグベースのアクセス許可を適用できるように、リソースにタグを割り当てることもできます。例えば、次のタグを使用できます。

- 特定のチーム、部門、またはアプリケーションが使用するリソースを追跡するためにタグを付けます。
- 機密文書など、特別な処理が必要なリソースにタグを付けます。
- スクリプトを使用してタグ付けされたリソースを自動的に管理し、ピーク時に使用を停止します。

以下のセクションでは、 テンプレートとタグ付けされたリソースを使用した AWS CloudFormation リソースの管理に関する追加情報を提供します。

トピック

- AWS CloudFormationでの X-Ray リソースの作成
- X-Ray のサンプリングルールとグループのタグ付け

AWS CloudFormationでの X-Ray リソースの作成

AWS X-Ray はAWS CloudFormation、 AWS リソースとインフラストラクチャの作成と管理に費やす時間を短縮できるように、リソースのモデル化とセットアップに役立つサービスである と統合されています。使用するすべての AWS リソースを記述するテンプレートを作成し、それらのリソースを AWS CloudFormation プロビジョニングして設定します。

を使用すると AWS CloudFormation、テンプレートを再利用して X-Ray リソースを一貫して繰り返しセットアップできます。リソースを 1 回記述し、同じリソースを複数の AWS アカウント およびリージョンで何度もプロビジョニングします。

X-Ray と AWS CloudFormation テンプレート

X-Ray および関連サービスのリソースをプロビジョニングおよび設定するには、AWS CloudFormation テンプレート を使用します。テンプレートは、JSON または YAML でフォーマットされたテキストファイルです。これらのテンプレートは、 AWS CloudFormation スタックでプロビジョニングするリソースを記述します。JSON または YAML に慣れていない場合は、 AWS CloudFormation デザイナー を使用してテンプレートの使用を開始 AWS CloudFormation できます。詳細については、「AWS CloudFormation ユーザーガイド」の「AWS CloudFormation Designer とは」を参照してください。

X-Ray は、 での AWS::XRay::Group、AWS::XRay::SamplingRule、および AWS::XRay::ResourcePolicyリソースの作成をサポートしています AWS CloudFormation。JSON テンプレートと YAML テンプレートの例を含む詳細については、AWS CloudFormation ユーザーガイドの「X-Ray リソースタイプのリファレンス」を参照してください。

の詳細 AWS CloudFormation

の詳細については AWS CloudFormation、以下のリソースを参照してください。

- AWS CloudFormation
- AWS CloudFormation ユーザーガイド
- AWS CloudFormation API リファレンス
- AWS CloudFormation コマンドラインインターフェイスユーザーガイド

X-Ray のサンプリングルールとグループのタグ付け

タグは、 AWS リソースを識別して整理するために使用できる単語またはフレーズです。各 リソースに複数のタグを追加できます。各タグには、ユーザーが定義するキーとオプションの値が含まれます。たとえば、タグキーは **domain**、タグ値は **example.com** などです。追加したタグに基づいて、リソースを検索したりフィルタ処理したりできます。タグの使用方法の詳細については、AWS 一般的なリファレンスの「タグ付け AWS リソース」を参照してください。

タグを使用して、 CloudFront ディストリビューションにタグベースのアクセス許可を適用できます。詳細については、「リソースタグを使用した AWS リソースへのアクセスの制御」を参照してください。

> **Note**
>
> タグエディタおよびAWS リソースグループは現在、X-Ray リソースをサポートしていません。タグを追加および管理するには、 AWS X-Ray コンソールまたは API を使用します。

X-Ray コンソール、API、 SDKs、および を使用して AWS CLI、リソースにタグを適用できます AWS Tools for Windows PowerShell。詳細については、次のドキュメントを参照してください。

- X-Ray API — AWS X-Ray API リファレンスの以下の操作を参照してください。
  - ListTagsForResource
  - CreateSamplingRule
  - CreateGroup
  - TagResource
  - UntagResource
- AWS CLI – コマンドリファレンスの「Xray」を参照してください。 AWS CLI
- SDK – AWS ドキュメントページの該当する SDK ドキュメントを参照

> **Note**
>
> X-Ray リソースでタグを追加または変更できない場合、または特定のタグを持つリソースを追加できない場合は、この操作を実行する権限がない可能性があります。アクセスをリクエ

ストするには、X-Ray で管理者権限を持つエンタープライズの AWS ユーザーにお問い合わせください。

トピック

- タグの制限
- コンソールでのタグの管理
- でのタグの管理 AWS CLI
- タグに基づいて X-Ray リソースへのアクセスを制御する

タグの制限

タグには次の制限があります。

- リソースあたりのタグの最大数 – 50
- キーの最大長 – 128 文字 (Unicode)
- 値の最大長 – 256 文字 (Unicode)
- キーと値の有効な値 – a ~ z、A ~ Z、0 ~ 9、スペース、特殊文字 (_ . : / = + - @)
- タグのキーと値では、大文字と小文字が区別されます。
- `aws:` をキーのプレフィックスとしてを使用しないでください。 AWS 用に予約済みです。

> **Note**
>
> システムタグを編集または削除することはできません。

コンソールでのタグの管理

X-Ray グループまたはサンプリングルールの作成時に、オプションのタグを追加できます。タグは、後でコンソールで変更または削除することもできます。

次の手順では、X-Ray コンソールでグループおよびサンプリングルールのタグを追加、編集、削除する方法について説明します。

トピック

- 新しい グループにタグを追加する (コンソール)
- 新しいサンプリングルールにタグを追加する (コンソール)
- グループのタグを編集または削除する (コンソール)
- サンプリングルールのタグを編集または削除する (コンソール)

新しい グループにタグを追加する (コンソール)

新しい X-Ray グループを作成するときに、オプションのタグをグループの作成ページで追加できます。

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/xray/home で X-Ray コンソールを開きます。
2. ナビゲーションペインで、設定を展開し、Groupsを選択します。
3. [グループを作成] を選択します。
4. グループの作成ページで、グループの名前とフィルタ式を指定します。これらのプロパティの詳細については、「グループを設定する」を参照してください。
5. タグで、タグキー、およびオプションでタグ値を入力します。たとえば、**Stage**のタグキー、および**Production**のタグ値を入力して、このグループが生産用途であることを示します。タグを追加すると、必要に応じて別のタグを追加するための新しい行が表示されます。タグの制限については、このトピックで タグの制限 を参照してください。
6. タグの追加が完了したら、[Create group (グループの作成)] を選択します。

新しいサンプリングルールにタグを追加する (コンソール)

新しい X-Ray のサンプリングルールを作成する際、サンプリングルールの作成ページでタグを追加することができます。

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/xray/home で X-Ray コンソールを開きます。
2. ナビゲーションペインで、設定を展開し、サンプリングを選択します。
3. サンプリングールの作成を選択します。
4. サンプリングールの作成ページで、名前、優先度、制限、一致条件、および一致する属性を指定します。これらのプロパティの詳細については、「サンプリングルールを設定する」を参照してください。

5. タグで、タグキー、およびオプションでタグ値を入力します。たとえば、**Stage**のタグキー、および**Production**のタグ値を入力して、このサンプリングルールが本番用途であることを示します。タグを追加すると、必要に応じて別のタグを追加するための新しい行が表示されます。タグの制限については、このトピックで タグの制限 を参照してください。
6. タグの追加が完了したら、[サンプリングルールの作成] を選択します。

グループのタグを編集または削除する (コンソール)

X-Ray グループのタグをグループの編集ページで変更または削除できます。

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/xray/home で X-Ray コンソールを開きます。
2. ナビゲーションペインで、設定を展開し、Groupsを選択します。
3. Groups テーブルで、グループの名前を選択します。
4. グループの編集ページ、タグで、タグキーと値を編集します。重複するタグキーを使用することはできません。タグ値はオプションです。必要に応じて値を削除できます。グループの編集ページでの他のプロパティの詳細については、「グループを設定する」を参照してください。タグの制限については、このトピックで タグの制限 を参照してください。
5. タグを削除するには、タグの右側にあるXを選択します。
6. タグの編集または削除を完了したら、[グループの更新] を選択します。

サンプリングルールのタグを編集または削除する (コンソール)

X-Ray サンプリングルールのタグをサンプリングルールの編集ページで変更または削除できます。

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/xray/home で X-Ray コンソールを開きます。
2. ナビゲーションペインで、設定を展開し、サンプリングを選択します。
3. サンプリングルールテーブルで、サンプリングルールの名前を選択します。
4. タグで、タグキーと値を編集します。重複するタグキーを使用することはできません。タグ値はオプションです。必要に応じて値を削除できます。サンプリングルールの編集ページでの他のプロパティの詳細については、「サンプリングルールを設定する」を参照してください。タグの制限については、このトピックで タグの制限 を参照してください。
5. タグを削除するには、タグの右側にあるXを選択します。
6. タグの編集または削除を完了したら、[サンプリングルールの更新] を選択します。

でのタグの管理 AWS CLI

X-Ray グループまたはサンプリングルールの作成時に、タグを追加できます。を使用してタグ AWS CLI を作成および管理することもできます。既存のグループまたはサンプリングルールのタグを更新するには、 AWS X-Ray コンソール、または TagResourceまたは UntagResource APIsを使用します。

トピック

- 新しい X-Ray グループまたはサンプリングルールにタグを追加する (CLI)
- 既存のリソースにタグを追加する (CLI)
- リソースのタグを一覧表示する (CLI)
- リソースのタグを削除する (CLI)

新しい X-Ray グループまたはサンプリングルールにタグを追加する (CLI)

新しい X-Ray グループまたはサンプリングルールの作成時にオプションのタグを追加するには、次のいずれかのコマンドを使用します。

- 新しいグループにタグを追加するには、以下のコマンドを実行し、*group_name*をグループ名に、*mydomain.com*をサービスのエンドポイントに、*key_name*をタグキーに、オプションで、*#*をタグ値に置き換えてください。グループの作成方法の詳細については、「X-Ray API を使用したサンプリング、グループ、および暗号化設定の構成」を参照してください。

  ```
  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"}]
  ```

  次に例を示します。

  ```
  aws xray create-group \
     --group-name "AdminGroup" \
     --filter-expression "service(\"mydomain.com\") {fault OR error}" \
     --tags [{"Key": "Stage","Value": "Prod"},{"Key": "Department","Value": "QA"}]
  ```

- 新しいサンプリングルールにタグを追加するには、以下のコマンドを実行し、*key_name*をタグキーに、オプションで、*#*をタグ値に置き換えてください。このコマンドは、--sampling-ruleパラメータの値をJSON ファイルとして指定します。サンプリングルールの作成方法の詳細につい

ては、「X-Ray API を使用したサンプリング、グループ、および暗号化設定の構成」を参照してください。

```
aws xray create-sampling-rule \
   --cli-input-json file://file_name.json
```

--cli-input-json パラメーターで指定された JSON ファイル*file_name.json*の内容は以下の通りです。

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

コマンドの例を次に示します。

```
aws xray create-sampling-rule \
   --cli-input-json file://9000-base-scorekeep.json
```

--cli-input-json パラメーターで指定されたサンプル 9000-base-scorekeep.json ファイルの内容は以下の通りです。

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

既存のリソースにタグを追加する (CLI)

tag-resourceコマンドを実行して既存の X-Ray グループまたはサンプリングルールにタグを追加するこの方法は、update-group や update-sampling-rule を実行してタグを追加するよりも簡単な場合があります。

グループまたはサンプリングルールにタグを追加するには、次のコマンドを実行し、ARN をリソースの ARN に置き換え、追加したいタグのキーとオプションの値を指定します。

```
aws xray tag-resource \
   --resource-arn "ARN" \
```

```
   --tag-keys [{"Key":"key_name","Value":"value"}, {"Key":"key_name","Value":"value"}]
```

次に例を示します。

```
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"}]
```

リソースのタグを一覧表示する (CLI)

`list-tags-for-resource`コマンドを実行して、X-Ray グループまたはサンプリングルールのタグを一覧表示できます。

グループまたはサンプリングルールに関連付けられているタグを一覧表示するには、次のコマンドを実行し、ARN をリソースの ARN に置き換えます。

```
aws xray list-tags-for-resource \
   --resource-arn "ARN"
```

次に例を示します。

```
aws xray list-tags-for-resource \
   --resource-arn "arn:aws:xray:us-east-2:01234567890:group/AdminGroup"
```

リソースのタグを削除する (CLI)

`untag-resource`コマンドを実行して、X-Ray グループまたはサンプリングルールからタグを削除できます。

グループまたはサンプリングルールからタグを削除するには、次のコマンドを実行し、ARN をリソースの ARN に置き換え、削除したいタグのキーを指定します。

`untag-resource`コマンドでタグ全体のみを削除できます。タグ値を削除するには、X-Ray コンソールを使用するか、タグを削除し、同じキーで異なる値または空の値で新しいタグを追加します。

```
aws xray untag-resource \
   --resource-arn "ARN" \
   --tag-keys ["key_name","key_name"]
```

次に例を示します。

```
aws xray untag-resource \
   --resource-arn "arn:aws:xray:us-east-2:01234567890:group/group_name" \
   --tag-keys ["Stage","Department"]
```

タグに基づいて X-Ray リソースへのアクセスを制御する

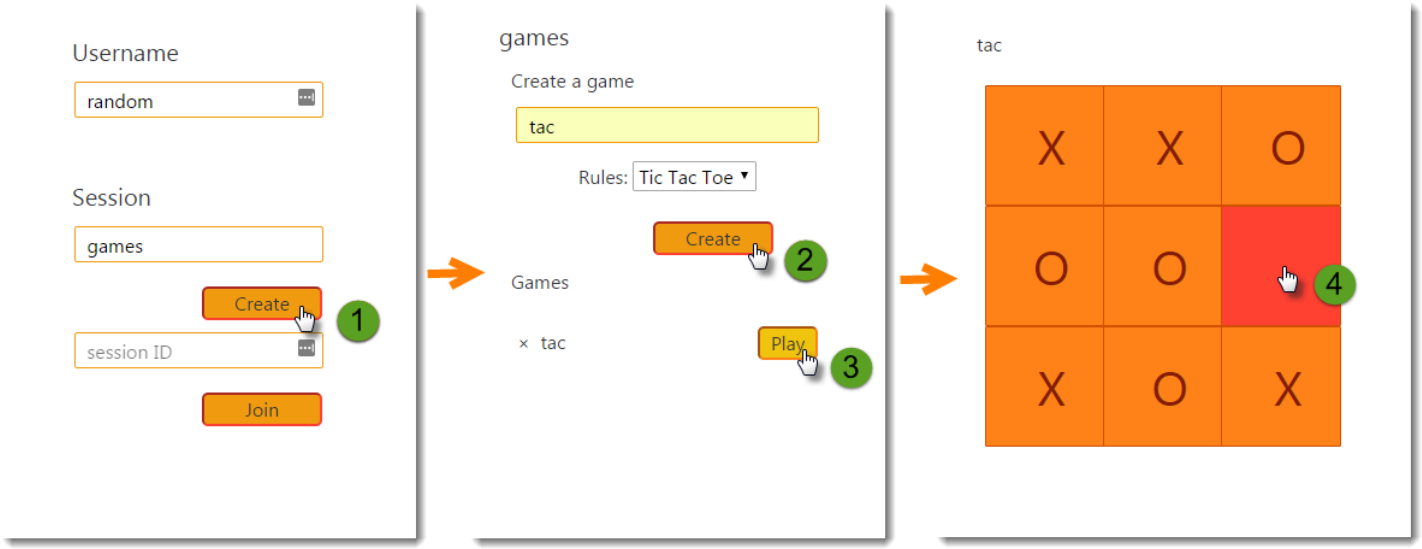
X-Ray グループまたはサンプリングルールにタグをアタッチしたり、リクエストでタグを X-Ray に渡すことができます。タグに基づいてアクセスを管理するには、`xray:ResourceTag/key-name`、`aws:RequestTag/key-name`、または `aws:TagKeys` の条件キーを使用して、ポリシーの条件要素でタグ情報を提供します。これらの条件キーの詳細については、AWS 「リソースタグ を使用したリソースへのアクセスの制御」を参照してください。

リソースのタグに基づいてリソースへのアクセスを制限するためのアイデンティティベースポリシーの例を表示するには、「タグに基づいて X-Ray グループおよびサンプリングルールへのアクセスを管理する」を参照してください。

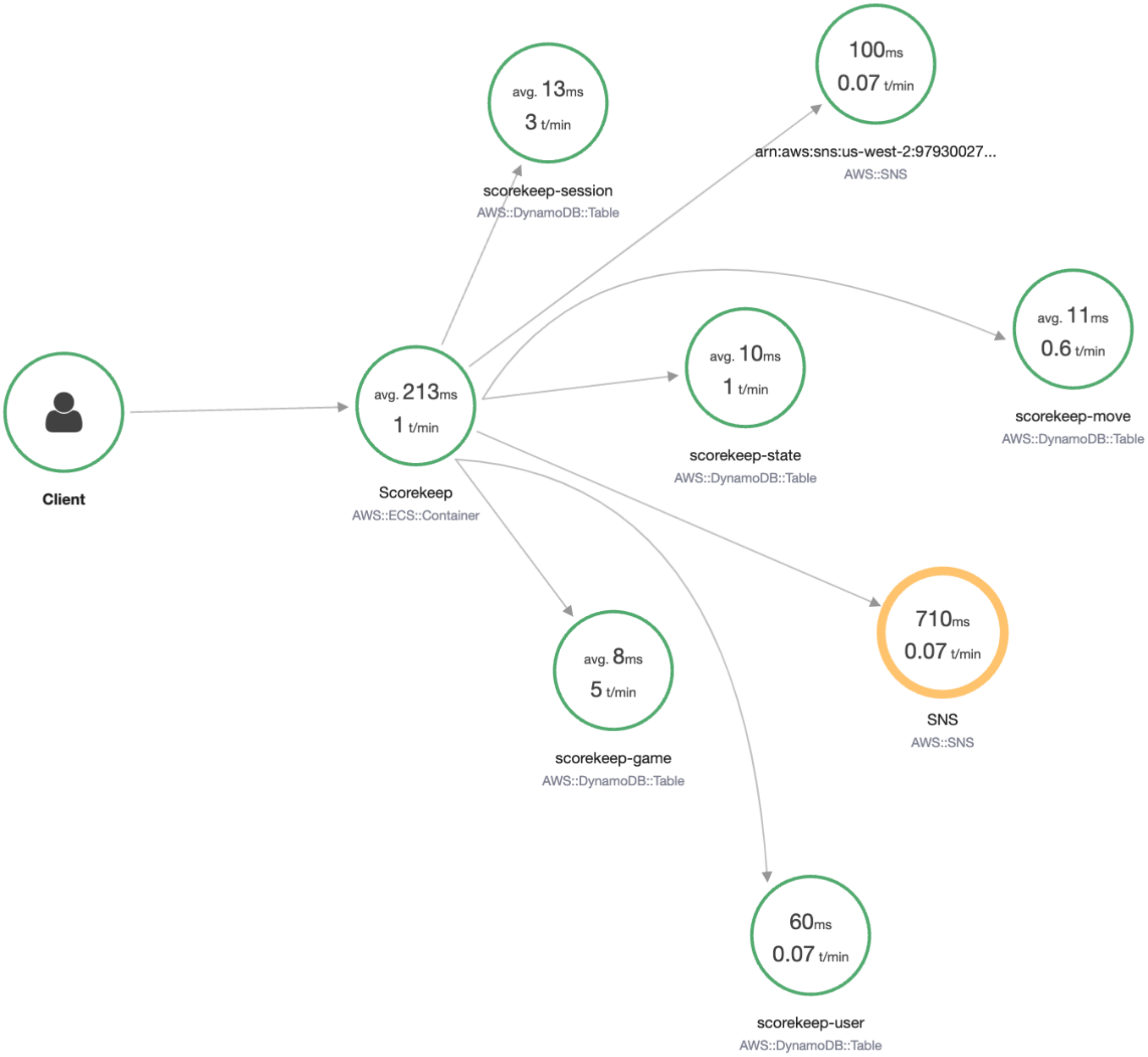
AWS X-Ray サンプルアプリケーション

で使用できる AWS X-Ray eb-java-scorekeep サンプルアプリケーションは GitHub、 AWS X-Ray SDK を使用して、受信 HTTP コール、DynamoDB SDK クライアント、および HTTP クライアントを計測する方法を示しています。サンプルアプリケーションは、 を使用して DynamoDB テーブル AWS CloudFormation を作成し、インスタンスで Java コードをコンパイルし、追加設定なしで X-Ray デーモンを実行します。

AWS Management Console または を使用して、計測されたサンプルアプリケーションのインストールと使用を開始するには、Scorekeep チュートリアルを参照してください AWS CLI。



サンプルには、フロントエンドのウェブアプリ、それが呼び出す API、データの保存に使用する DynamoDB テーブルが含まれています。フィルター 、プラグイン 、および計測された AWS SDK クライアントを使用した基本的な計測は、プロジェクトの `xray-gettingstarted`ブランチに表示されます。これは、入門ガイドチュートリアルでデプロイするブランチです。このブランチには基本情報しか含まれていないので、`master` ブランチと比較すると、基本をすばやく理解できます。



同じアプリケーションで、次のファイルの基本計測も説明します。

- HTTP リクエストフィルタ–WebConfig.java
- AWS SDK クライアントの計測 – build.gradle

アプリケーションの `xray` ブランチには、HTTPClient、注釈、SQL クエリ、カスタムサブセグメント、実装された AWS Lambda 関数、および実装された初期化コードとスクリプトの使用が含まれています。

ブラウザでのユーザーのログインと AWS SDK for JavaScript 使用をサポートするために、 `xray-cognito`ブランチは Amazon Cognito を追加してユーザーの認証と認可をサポートします。また、Amazon Cognitoから認証情報を取得すると、ウェブアプリケーションはトレースデータをX-Rayに送信してクライアントの観点からリクエスト情報を記録します。ブラウザクライアントはトレースマップに独自のノードとして表示され、ユーザーが表示しているページの URL やユーザーの ID などの追加情報を記録します。

最後に、 `xray-worker`ブランチは、個別に実行され、Amazon SQS キューの項目を処理する、実装された Python Lambda 関数を追加します。Scorekeep は、ゲームが終了するたびに項目をキューに追加します。 CloudWatch イベントによってトリガーされる Lambda ワーカーは、数分ごとにキューから項目を取得し、分析のために Amazon S3 にゲームレコードを保存するように処理します。

トピック

- Scorekeep サンプルアプリケーションの開始方法
- AWS SDK クライアントの手動計測
- 追加のサブセグメントを作成する
- 注釈、メタデータ、およびユーザー ID を記録する
- 送信 HTTP 呼び出しの計測
- PostgreSQL データベースに対する呼び出しの計測
- AWS Lambda 関数の計測
- スタートアップコードの作成
- 実装スクリプト
- ウェブアプリケーションクライアントの実装
- 実装されたクライアントをワーカースレッドで使用する

Scorekeep サンプルアプリケーションの開始方法

このチュートリアルでは、Scorekeep サンプルアプリケーション の `xray-gettingstarted`ブランチを使用します。Scorekeep サンプルアプリケーション は、 を使用して、Amazon ECS でサン

プルアプリケーションと X-Ray デーモンを実行するリソース AWS CloudFormation を作成および設定します。アプリケーションは Spring フレームワークを使用して JSON ウェブ API を実装し、AWS SDK for Java を使用してデータを Amazon DynamoDB に保持します。アプリケーションのサーブレットフィルターは、アプリケーションによって処理されるすべての受信リクエストを計測し、 AWS SDK クライアントのリクエストハンドラーは DynamoDB へのダウンストリーム呼び出しを計測します。

このチュートリアルは、 AWS Management Console または のいずれかを使用して実行できます AWS CLI。

セクション

前提条件

このチュートリアルでは AWS CloudFormation 、 を使用して、サンプルアプリケーションと X-Ray デーモンを実行するリソースを作成および設定します。チュートリアルでインストールと実行をする前提条件として以下が必要です。

1. アクセス許可が限定された IAM ユーザーを使用する場合は、IAM コンソールに次のユーザーポリシーを追加してください。

   - `AWSCloudFormationFullAccess` – にアクセスして使用する CloudFormation
   - `AmazonS3FullAccess` — を使用してテンプレートファイルを CloudFormation にアップロードする AWS Management Console
   - `IAMFullAccess` – Amazon ECS インスタンスロールと Amazon EC2 インスタンスロールを作成

- `AmazonEC2FullAccess` – Amazon EC2 リソースを作成
- `AmazonDynamoDBFullAccess` – DynamoDB テーブルを作成
- `AmazonECS_FullAccess` – Amazon ECS リソースを作成
- `AmazonSNSFullAccess` – Amazon SNS トピックを作成
- `AWSXrayReadOnlyAccess` - X-Ray コンソールでトレースマップとトレースを表示するアクセス許可用

2. を使用してチュートリアルを実行するには AWS CLI、 CLI バージョン 2.7.9 以降をインストールし、前のステップの ユーザーを使用して CLI を設定します。ユーザー AWS CLI で を設定するときは、リージョンが設定されていることを確認してください。リージョンが設定されていない場合は、すべての CLI コマンドに `--region` *`AWS-REGION`* を追加する必要があります。
3. サンプルアプリケーションリポジトリを複製ために、Git がインストールされていることを確認してください。
4. 次のコード例を使用して、Scorekeep リポジトリの `xray-gettingstarted`ブランチのクローンを作成します。

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

を使用して Scorekeep アプリケーションをインストールする CloudFormation

AWS Management Console

を使用してサンプルアプリケーションをインストールする AWS Management Console

1. CloudFormation コンソールを開きます。
2. [スタックの作成] を選択し、ドロップダウンメニューから [新しいリソースを使用] を選択します。
3. [テンプレートの指定] セクションで、[テンプレートファイルのアップロード] を選択します。
4. [ファイルの選択] を選択し、git リポジトリをクローンしたときに作成された `xray-scorekeep/cloudformation` フォルダーに移動して、`cf-resources.yaml` ファイルを選択します。
5. [次へ] を選択して続行します。

6. [スタック名] テキストボックスに scorekeep と入力し、ページ下部の [次へ] を選択して続行します。このチュートリアルのこれ以降の部分ではスタックの名前を scorekeep とします。
7. [スタックオプションの設定] ページの一番下までスクロールし、[次へ] を選択して続行します。
8. レビューページの下部までスクロールし、カスタム名で IAM リソースを作成する CloudFormation 可能性のあるチェックボックスを確認し、スタックの作成を選択します。
9. CloudFormation スタックは作成中です。スタックのステータスは約 5 分間 CREATE_COMPLETE で、その後 CREATE_IN_PROGRESS に変わります。ステータスは定期的に更新されます。ページを更新して再表示することもできます。

AWS CLI

を使用してサンプルアプリケーションをインストールする AWS CLI

1. このチュートリアルの前の部分でクローンを作成した xray-scorekeep リポジトリの cloudformation フォルダーに移動します。

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

2. 次の AWS CLI コマンドを入力してスタックを作成します CloudFormation 。

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

3. CloudFormation スタックのステータスが になるまで待ちCREATE_COMPLETEます。これには約 5 分かかります。ステータスを確認するには、次の AWS CLI コマンドを使用します。

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

トレースデータの生成

サンプルアプリケーションには、フロントエンドのウェブアプリケーションが含まれています。ウェブアプリケーションを使用して API へのトラフィックを生成し、トレースデータを X-Ray に送信します。まず、 AWS Management Console または AWS CLIを使用してウェブアプリの URL を取得します。

AWS Management Console

を使用してアプリケーション URL を検索する AWS Management Console

1. CloudFormation コンソールを開きます。
2. リストから scorekeep スタックを選択します。
3. scorekeep スタックページの [出力] タブを選択し、LoadBalancerUrl URL リンクを選択してウェブアプリケーションを開きます。

AWS CLI

を使用してアプリケーション URL を検索する AWS CLI

1. 次のコマンドを使用して、ウェブアプリケーションの URL を表示します。

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

2. この URL をコピーしてブラウザで開き、Scorekeep ウェブアプリケーションを表示します。

ウェブアプリケーションを使用してトレースデータを生成する

1. [Create] を選択して、ユーザーとセッションを作成します。
2. [game name] を入力し、[Rules] を [Tic Tac Toe] に設定したら、[Create] を選択して、ゲームを作成します。
3. [Play] を選択してゲームを開始します。
4. ゲームの状態を移行および変更するには、タイルを選択します。

これらの各ステップで、API への HTTP リクエスト、および DynamoDB へのダウンストリーム呼び出しが生成され、ユーザー、セッション、ゲーム、移動、および状態データが読み書きされます。

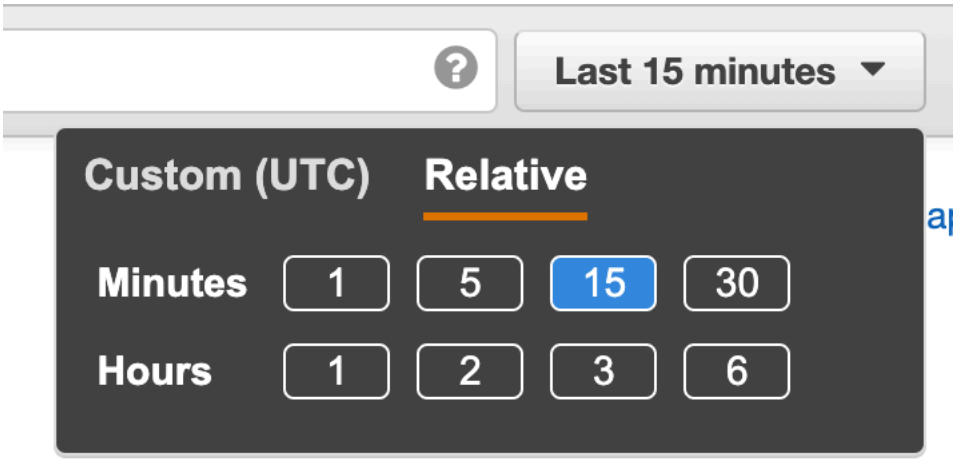
でトレースマップを表示する AWS Management Console

サンプルアプリケーションによって生成されたトレースマップとトレースは、X-Ray および CloudWatch コンソールで確認できます。

X-Ray console

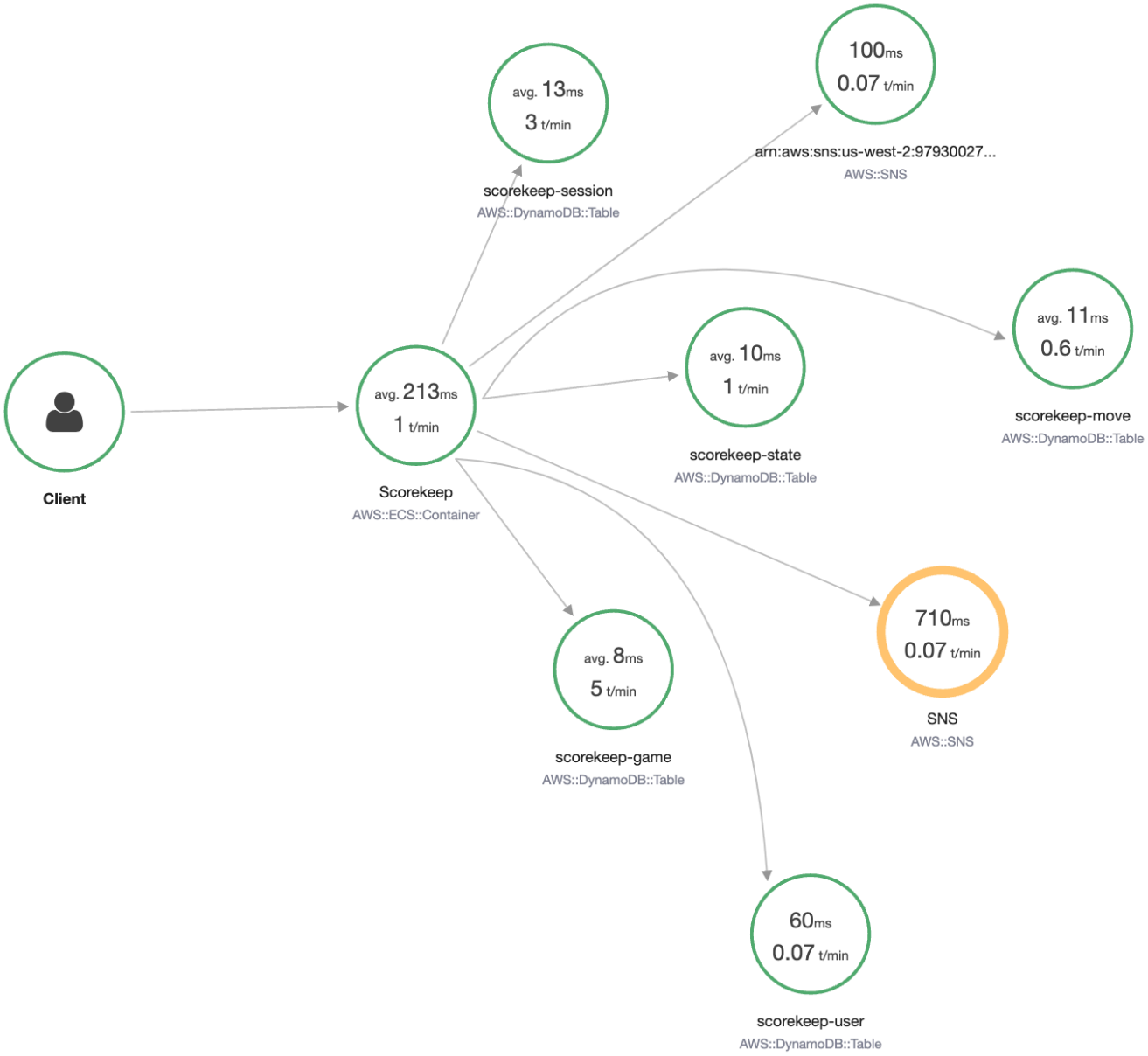
X-Ray コンソールを使用する

1. X-Ray コンソールのトレースマップページを開きます。
2. コンソールには、X-Ray によってアプリケーションから送信されたトレースデータから生成されたサービスグラフの表現が表示されます。必要に応じてトレースマップの期間を調整し、ウェブアプリケーションを最初に起動してからすべてのトレースが表示されることを確認します。



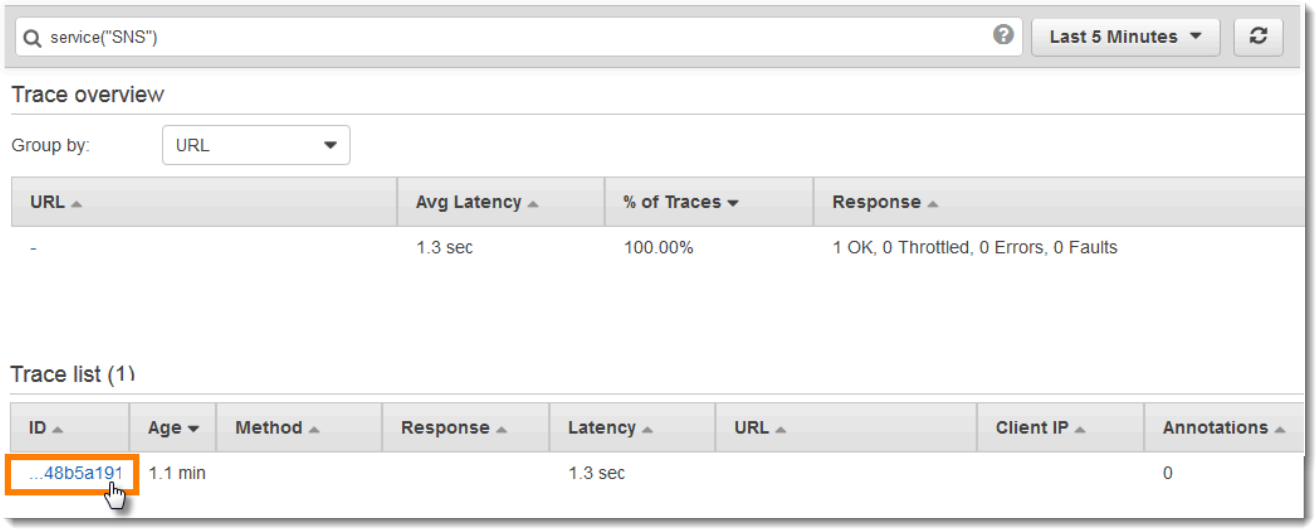
トレースマップには、ウェブアプリケーションクライアント、Amazon ECS で実行されている API、およびアプリケーションが使用する各 DynamoDB テーブルが表示されます。アプリケーションに対するすべてのリクエストは、1 秒あたりのリクエストの設定可能な最大数まで、API にヒットした際にトレースされ、ダウンストリームサービスへのリクエストを生成して、完了します。

サービスグラフの任意のノードを選択すると、そのノードに対してトラフィックを生成したリクエストのトレースを表示できます。現在、Amazon SNS ノードは黄色になっています。理由を調べるために掘り下げます。

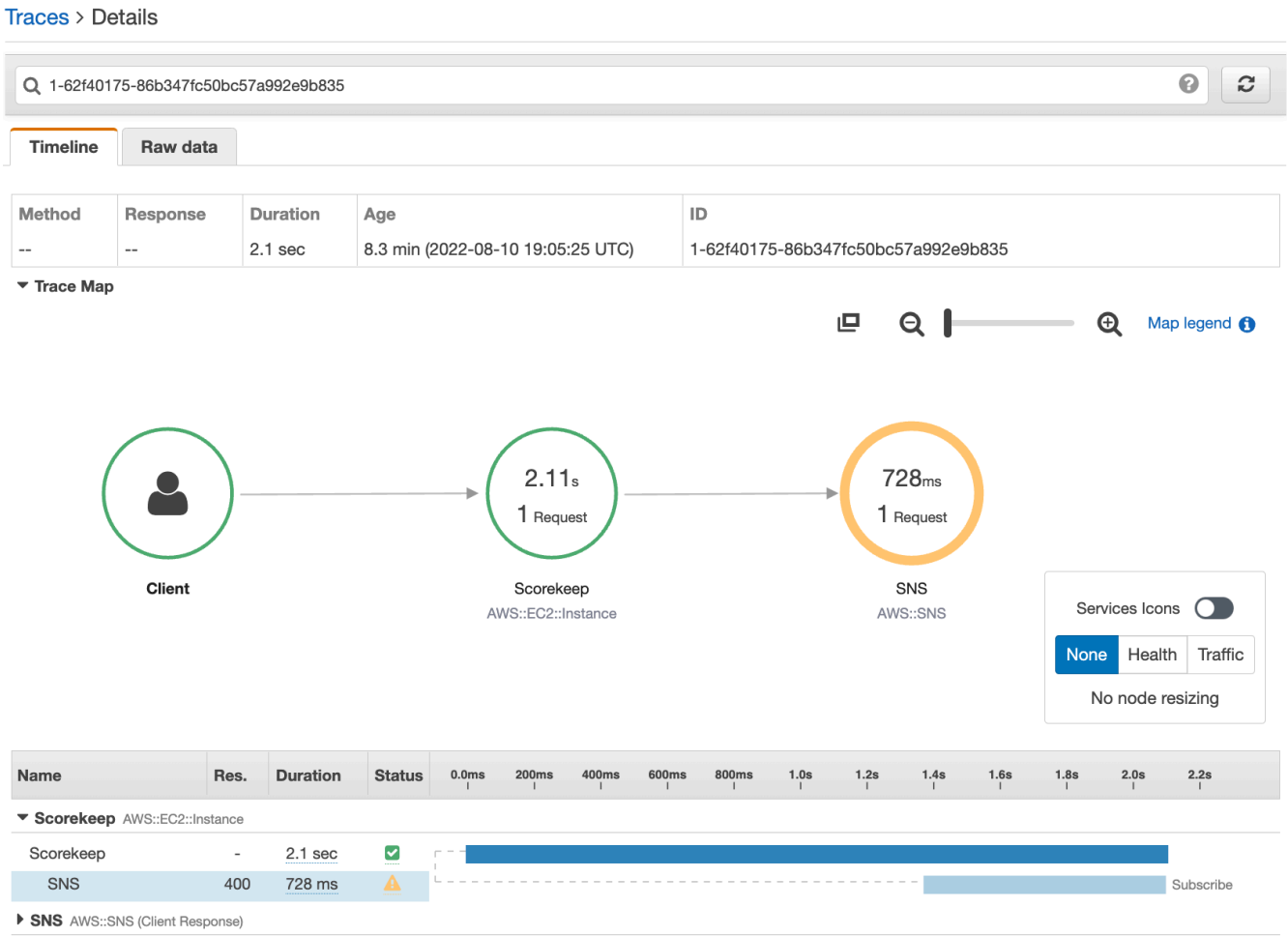


エラーの原因を見つけるには

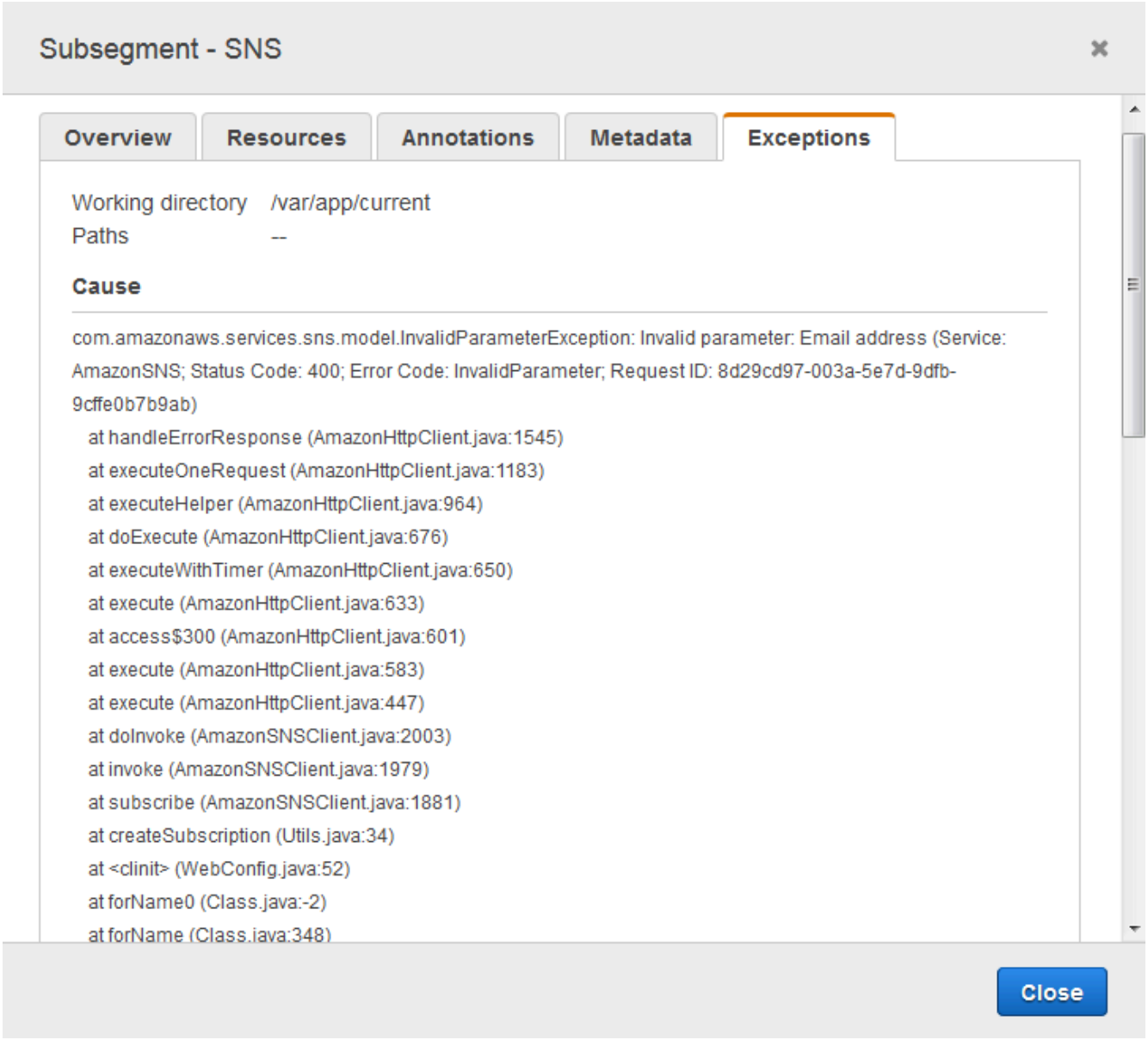
1. [SNS] という名前のノードを選択します。ノードの詳細パネルが表示されます。
2. [トレースの表示] を選択して、[トレースの概要] 画面にアクセスします。
3. [Trace のリスト] からトレースを選択します。受信リクエストに応答するのではなく起動時に記録されているため、トレースには、メソッドまたは URL はありません。



4. ページ下部の Amazon SNS セグメント内のエラーステータスアイコンを選択し、SNS サブセグメントの [例外] ページを開きます。



5. X-Ray SDK は、計測された AWS SDK クライアントによってスローされた例外を自動的にキャプチャし、スタックトレースを記録します。

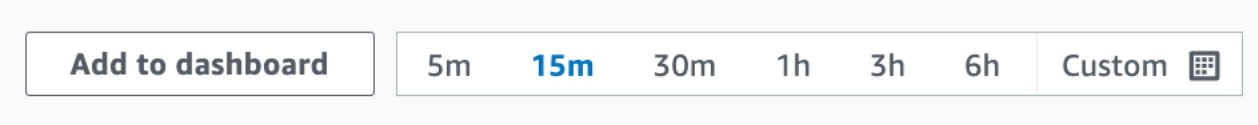


CloudWatch console

CloudWatch コンソールを使用する

1. CloudWatch コンソールの X-Ray トレースマップページを開きます。
2. コンソールには、X-Ray によってアプリケーションから送信されたトレースデータから生成されたサービスグラフの表現が表示されます。必要に応じてトレースマップの期間を調整

し、ウェブアプリケーションを最初に起動してからすべてのトレースが表示されることを確認します。



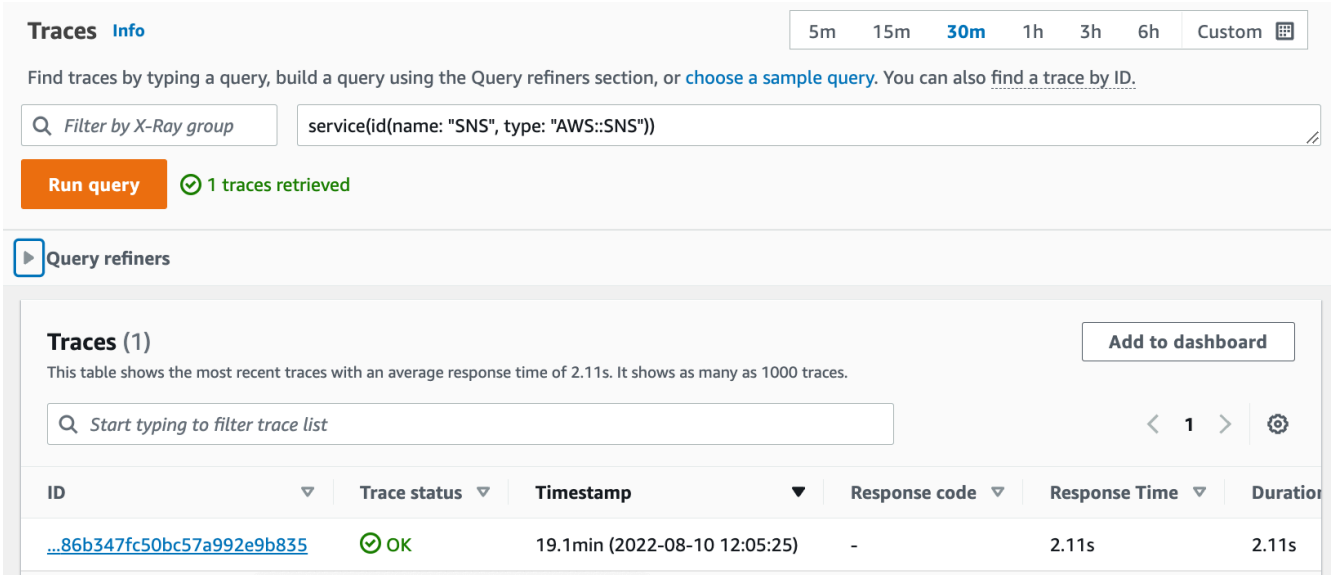
トレースマップには、ウェブアプリケーションクライアント、Amazon EC2 で実行されている API、およびアプリケーションが使用する各 DynamoDB テーブルが表示されます。アプリケーションに対するすべてのリクエストは、1 秒あたりのリクエストの設定可能な最大数まで、API にヒットした際にトレースされ、ダウンストリームサービスへのリクエストを生成して、完了します。

サービスグラフの任意のノードを選択すると、そのノードに対してトラフィックを生成したリクエストのトレースを表示できます。現在、Amazon SNS ノードはオレンジ色になっています。理由を調べるために掘り下げます。

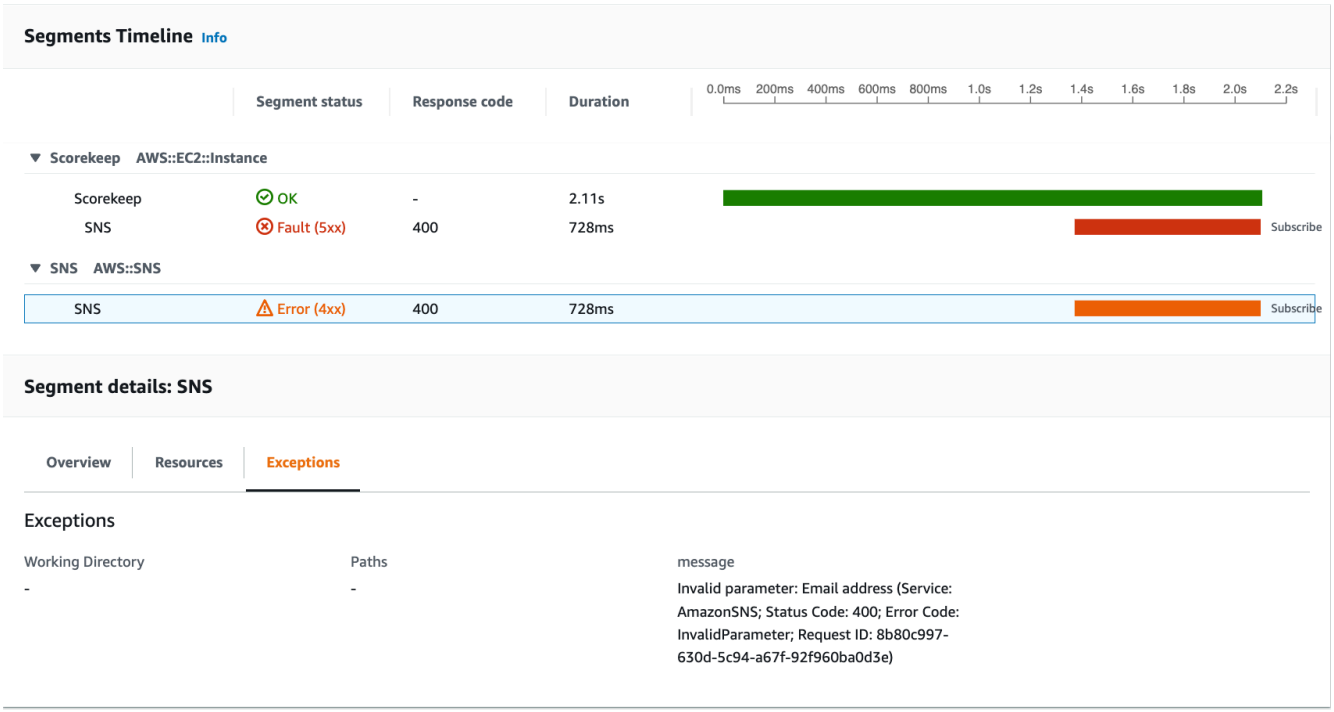

scorekeep-state
DynamoDB Table
scorekeep-game
DynamoDB Table
scorekeep-move
DynamoDB Table
Client
Scorekeep
ECS Container
SNS
SNS
scorekeep-session
DynamoDB Table

エラーの原因を見つけるには

1. [SNS] という名前のノードを選択します。SNS ノードの詳細パネルがマップの下に表示されます。
2. [トレースを表示] を選択して [トレース] ページにアクセスします。
3. ページの下部で、[トレース] リストからトレースを選択します。受信リクエストに応答するのではなく起動時に記録されているため、トレースには、メソッドまたは URL はありません。



4. セグメントタイムラインの下部にある Amazon SNS サブセグメントを選択し、SNS サブセグメントの [例外] タブを選択すると、例外の詳細が表示されます。



この原因は、`WebConfig` クラスで行われた `createSubscription` の呼び出しで指定された E メールアドレスが無効であることを意味します。次のセクションで、これを修正します。

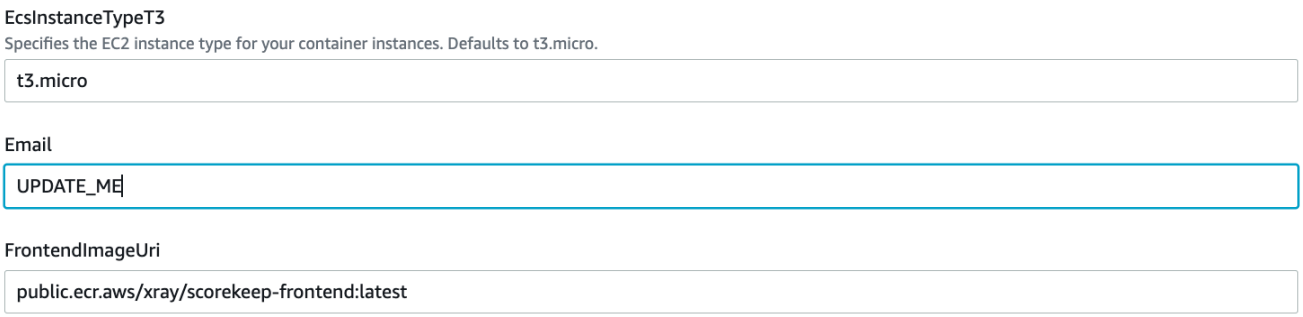
Amazon SNS 通知の設定

Scorekeep では、ユーザーがゲームを達成すると、Amazon SNS で通知が送信されます。アプリケーションが起動すると、 CloudFormation スタックパラメータで定義された E メールアドレスのサブスクリプションを作成しようとします。現在その呼び出しに失敗しています。通知を有効にするように通知 E メールを設定し、トレースマップで強調表示されている障害を解決します。

AWS Management Console

を使用して Amazon SNS 通知を設定するには AWS Management Console

1. CloudFormation コンソールを開きます。
2. リストの `scorekeep` スタック名の横にあるラジオボタンを選択して、[更新] を選択します。
3. [現在のテンプレートの使用] が選択されていることを確認し、[スタックの更新] ページで [次へ] をクリックします。
4. リストから [E メール] パラメータを探し、デフォルト値を有効な E メールアドレスに置き換えます。



5. ページの下部にスクロールし、[次へ] を選択します。
6. レビューページの下部までスクロールし、カスタム名で IAM リソースを作成する CloudFormation 可能性のあるチェックボックスを確認し、スタックの更新を選択します。
7. CloudFormation スタックは更新中です。スタックのステータスは約 5 分間 UPDATE_COMPLETE で、その後 UPDATE_IN_PROGRESS に変わります。ステータスは定期的に更新されます。ページを更新して再表示することもできます。

AWS CLI

を使用して Amazon SNS 通知を設定するには AWS CLI

1. 以前に作成した `xray-scorekeep/cloudformation/` フォルダーに移動し、`cf-resources.yaml` ファイルをテキストエディタで開きます。
2. [E メール] パラメータ内の `Default` 値を検索し、*`UPDATE_ME`* から有効な E メールアドレスに変更します。

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

3. `cloudformation` フォルダから、次の AWS CLI コマンドを使用して CloudFormation スタックを更新します。

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

4. CloudFormation スタックのステータスが になるまで待ちUPDATE_COMPLETEます。これには数分かかります。ステータスを確認するには、次の AWS CLI コマンドを使用します。

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

更新が完了すると、Scorekeep が再起動し、SNS トピックへのサブスクリプションが作成されます。E メールとサブスクリプションを確認して、ゲーム達成時にアップデートの有無を確認します。トレースマップを開いて、SNS への呼び出しが失敗していないことを確認します。

サンプルアプリケーションの詳細

サンプルアプリケーションは、X-Ray SDK for Java を使用するように設定された Java の HTTP ウェブ API です。 CloudFormation テンプレートを使用してアプリケーションをデプロイすると、DynamoDB テーブル、Amazon ECS クラスター、および ECS で Scorekeep を実行するために必要なその他のサービスが作成されます。ECS のタスク定義ファイルは、 を通じて作成されます CloudFormation。このファイルは ECS クラスター内のタスクごとに使用されるコンテナイメージを定義します。これらのイメージは、公式の X-Ray パブリック ECR から取得されます。Scorekeep API コンテナイメージには Gradle でコンパイルされた API が含まれています。Scorekeep フロントエンドコンテナのコンテナイメージは、nginx プロキシサーバーを使用するフロントエンドに対応します。このサーバーは /api で始まるパスにリクエストをルーティングして API に送信します。

受信 HTTP リクエストを測定するには、アプリケーションで SDK によって提供された `TracingFilter` を追加します。

Example src/main/java/scorekeep/WebConfig.java - サーブレットフィルター

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

@Configuration
public class WebConfig {

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

このフィルタは、アプリケーションが処理するすべての受信リクエストに関するトレースデータを送信します。リクエスト URL、メソッド、レスポンスステータス、開始時間、終了時間が含まれます。

また、アプリケーションは AWS SDK for Javaを使用して DynamoDB に対するダウンストリーム呼び出しを行います。これらの呼び出しを計測するために、アプリケーションは AWS SDK 関連のサブモジュールを依存関係として受け取るだけで、X-Ray SDK for Java はすべての AWS SDK クライアントを自動的に計測します。

アプリケーションは `Docker` を使用して、インスタンス上で `Gradle Docker Image` と `Scorekeep API Dockerfile` ファイルを使用するソースコードを構築し、Gradle の `ENTRYPOINT` で生成する実行可能 JAR を実行します。

Example Docker を使用して Gradle Docker イメージ経由で構築する

```
docker run --rm -v /PATH/TO/SCOREKEEP_REPO/home/gradle/project -w /home/gradle/project
 gradle:4.3 gradle build
```

Example Dockerファイル ENTRYPOINT

```
ENTRYPOINT [ "sh", "-c", "java -Dserver.port=5000 -jar scorekeep-api-1.0.0.jar" ]
```

SDK サブモジュールを依存関係として宣言することで、コンパイル中に `build.gradle` ファイルによって SDK サブモジュールが Maven からダウンロードされます。

Example build.gradle -- 依存関係

```
...
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")
    }
```

```
}
```

core、 AWS SDK、および AWS SDK Instrumentor サブモジュールは、 AWS SDK で行われたダウンストリーム呼び出しを自動的に計測するために必要なすべてです。

未加工のセグメントデータを X-Ray API に中継するには、X-Ray デーモンが UDP ポート 2000 でトラフィックを受信する必要があります。そのため、アプリケーションでは、ECS で Scorekeep アプリケーションとともにサイドカーコンテナとしてデプロイされるコンテナでX-Ray デーモンを実行します。詳細については、X-Ray デーモンのトピックを参照してください。

Example ECS タスク定義内の X-Ray デーモンコンテナ定義

```
...
Resources:
  ScorekeepTaskDefinition:
    Type: AWS::ECS::TaskDefinition
    Properties:
      ContainerDefinitions:
      ...

      - Cpu: '256'
        Essential: true
        Image: amazon/aws-xray-daemon
        MemoryReservation: '128'
        Name: xray-daemon
        PortMappings:
          - ContainerPort: '2000'
            HostPort: '2000'
            Protocol: udp
      ...
```

X-Ray SDK for Java は、AWSXRay という名前のクラスを提供します。これはコードを計測するために使用する TracingHandler というグローバルレコーダーを提供します。グローバルレコーダーを設定して、受信 HTTP 呼び出しのセグメントを作成する AWSXRayServletFilter をカスタマイズできます。サンプルには、プラグインとサンプリングルールでグローバルレコーダーを設定する WebConfig クラスの静的ブロックが含まれています。

Example src/main/java/scorekeep/WebConfig.java - レコーダー

```
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());
    ...

  }
}
```

この例では、ビルダーを使用して sampling-rules.json という名前のファイルからサンプリングルールをロードします。サンプリングルールは、SDK が受信リクエストのセグメントを記録するレートを決定します。

Example src/main/java/resources/sampling-rules.json

```
{
  "version": 1,
  "rules": [
    {
      "description": "Resource creation.",
      "service_name": "*",
      "http_method": "POST",
      "url_path": "/api/*",
      "fixed_target": 1,
      "rate": 1.0
    },
    {
      "description": "Session polling.",
      "service_name": "*",
      "http_method": "GET",
```

```
      "url_path": "/api/session/*",
      "fixed_target": 0,
      "rate": 0.05
    },
    {
      "description": "Game polling.",
      "service_name": "*",
      "http_method": "GET",
      "url_path": "/api/game/*/*",
      "fixed_target": 0,
      "rate": 0.05
    },
    {
      "description": "State polling.",
      "service_name": "*",
      "http_method": "GET",
      "url_path": "/api/state/*/*/*",
      "fixed_target": 0,
      "rate": 0.05
    }
  ],
  "default": {
    "fixed_target": 1,
    "rate": 0.1
  }
}
```

サンプリングルールファイルでは、4 つのカスタムサンプリングルールおよびデフォルトルールが定義されます。各受信リクエスト用に、SDK は定義された順にカスタムルールを適用します。SDK は、リクエストのメソッド、パス、サービス名に一致する最初のルールを適用します。Scorekeep の場合、最初のルールは、1 秒あたり 1 リクエストの固定ターゲット、および固定ターゲットが満たされた後の 1.0 または 100% のリクエストのレートを適用して、すべての POST リクエスト (リソース作成呼び出し) をキャッチします。

他の 3 つのカスタムルールでは、固定ターゲットなしで、セッション、ゲーム、および状態の読み取り (GET リクエスト) に 5% のレートを適用します。これにより、フロントエンドが、コンテンツが最新であることを確認するために数秒ごとに自動的に行う周期的呼び出しのトレース数を最小限に抑えることができます。他のすべてのリクエストの場合は、ファイルは、1 秒あたり 1 リクエストのデフォルトレートおよび 10% のレートを定義します。

また、サンプルアプリケーションでは、手動 SDK クライアント計測、追加サブセグメントの作成、HTTP 呼び出しの出力など、高度な機能の使用方法も説明します。詳細については、「AWS X-Ray サンプルアプリケーション」を参照してください。

オプション: 最小特権ポリシー

Scorekeep ECS コンテナは、AmazonSNSFullAccess や AmazonDynamoDBFullAccess などのフルアクセスポリシーを使用してリソースにアクセスします。フルアクセスポリシーの使用は、本稼働アプリケーションではベストプラクティスではありません。次の例では、DynamoDB IAM ポリシーを更新してアプリケーションのセキュリティを向上させます。IAM ポリシーのセキュリティのベストプラクティスの詳細については、AWS 「X-Ray の Identity and Access Management」を参照してください。

Example cf-resources.yaml テンプレート ECS TaskRole 定義

```
ECSTaskRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
          -
            Effect: "Allow"
            Principal:
              Service:
                - "ecs-tasks.amazonaws.com"
            Action:
              - "sts:AssumeRole"
      ManagedPolicyArns:
        - "arn:aws:iam::aws:policy/AmazonDynamoDBFullAccess"
        - "arn:aws:iam::aws:policy/AmazonSNSFullAccess"
        - "arn:aws:iam::aws:policy/AWSXrayFullAccess"
      RoleName: "scorekeepRole"
```

ポリシーを更新するには、最初に DynamoDB リソースの ARN を特定します。次に、カスタム IAM ポリシーで ARN を使用します。最後に、そのポリシーをインスタンスプロファイルに適用します。

DynamoDB リソースの ARN を識別するには :

1. DynamoDB コンソールを開きます。
2. 左側のナビゲーションバーから [テーブル] を選択します。

3. scorekeep-* のいずれかを選択すると、テーブルの詳細ページが表示されます。
4. [概要] タブで [追加情報] を選択してセクションを展開し、Amazon リソースネーム (ARN) を表示します。この値をコピーします。
5. AWS_REGION および AWS_ACCOUNT_ID の値を特定のリージョンとアカウント ID に置き換えて、ARN を次の IAM ポリシーに挿入します。この新しいポリシーは、すべてのアクションを許可する AmazonDynamoDBFullAccess ポリシーではなく、指定されたアクションのみ許可します。

   Example

   ```
   {
       "Version": "2012-10-17",
       "Statement": [
           {
               "Sid": "ScorekeepDynamoDB",
               "Effect": "Allow",
               "Action": [
                   "dynamodb:PutItem",
                   "dynamodb:UpdateItem",
                   "dynamodb:DeleteItem",
                   "dynamodb:GetItem",
                   "dynamodb:Scan",
                   "dynamodb:Query"
               ],
               "Resource": "arn:aws:dynamodb:<AWS_REGION>:<AWS_ACCOUNT_ID>:table/
   scorekeep-*"
           }
       ]
   }
   ```

   アプリケーションが作成するテーブルは、一貫した命名規則に従います。scorekeep-* 形式を使用して、すべての Scorekeep テーブルを指定できます。

IAM ポリシーを変更する

1. IAM コンソールから Scorekeep タスクロール (scorekeepRole) を開きます。
2. AmazonDynamoDBFullAccess ポリシーの横にあるチェックボックスを選択して [削除] を選択し、このポリシーを削除します。
3. [アクセス許可の追加]、[ポリシーのアタッチ]、[ポリシーの作成] の順に選択します。

4. [JSON] タブを選択し、上で作成したポリシーを貼り付けます。
5. ページ下部の [次へ: タグ] を選択します。
6. ページ下部の [次へ: 確認] を選択します。
7. [名前] に、ポリシーの名前を割り当てます。
8. ページの下部の [ポリシーの作成] を選択します。
9. 新しく作成したポリシーを `scorekeepRole` ロールにアタッチします。アタッチしたポリシーが適用されるまで数分かかることがあります。

新しいポリシーを`scorekeepRole`ロールにアタッチした場合は、 CloudFormation スタックを削除する前にデタッチする必要があります。このアタッチされたポリシーにより、スタックの削除がブロックされるためです。ポリシーを削除すると、ポリシーを自動的にデタッチできます。

カスタム IAM ポリシーを削除する

1. [IAM コンソール] を開きます。
2. 左側のナビゲーションバーから [ポリシー] を選択します。
3. このセクションで先ほど作成したカスタムポリシー名を検索し、ポリシー名の横にあるラジオボタンを選択して強調表示します。
4. [アクション] ドロップダウンを選択してから、[削除] を選択します。
5. カスタムポリシーの名前を入力して [削除] を選択し、削除を確定します。これにより、ポリシーが `scorekeepRole` ロールから自動的にデタッチされます。

クリーンアップ

Scorekeep アプリケーションのリソースを削除するには、次の手順を従います。

> **Note**
>
> このチュートリアルの前のセクションを使用してカスタムポリシーを作成してアタッチした場合は、 CloudFormation スタックを削除する`scorekeepRole`前に からポリシーを削除する必要があります。

AWS Management Console

を使用してサンプルアプリケーションを削除する AWS Management Console

1. CloudFormation コンソールを開きます。
2. リストの scorekeep スタック名の横にあるラジオボタンを選択して、[削除] を選択します。
3. CloudFormation スタックは削除中です。スタックのステータスは、すべてのリソースが削除されるまで数分間 DELETE_IN_PROGRESS になります。ステータスは定期的に更新されます。ページを更新して再表示することもできます。

AWS CLI

を使用してサンプルアプリケーションを削除する AWS CLI

1. 次の AWS CLI コマンドを入力して、 CloudFormation スタックを削除します。

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

2. CloudFormation スタックが存在しなくなるまで待ちます。これには約 5 分かかります。ステータスを確認するには、次の AWS CLI コマンドを使用します。

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

次のステップ

次の章「概念」で X-Ray の詳細をご覧ください。。

独自のアプリケーションを測定するには、X-Ray SDK for Javaまたは他の X-Ray SDK のいずれかの詳細をご覧ください。

- X-Ray SDK for Java – AWS X-Ray の SDK Java
- X-Ray SDK for Node.js – AWS Node.js 用 X-Ray SDK
- X-Ray SDK for .NET – AWS X-Ray SDK for .NET

X-Ray デーモンをローカルまたは で実行するには AWS、「」を参照してくださいAWS X-Ray デーモン。

でサンプルアプリケーションに寄稿するには GitHub、「」を参照してくださいeb-java-scorekeep。

AWS SDK クライアントの手動計測

X-Ray SDK for Java は、ビルド依存関係 に AWS SDK Instrumentor サブモジュールを含めると、すべての SDK クライアントを自動的に計測します。 AWS

クライアントの自動実装は、Instrumentor サブモジュールを削除することで無効にできます。これにより、他を無視しながら一部のクライアントを手動で実装するか、異なるクライアントで異なるトレースハンドラーを使用できるようになります。

特定の AWS SDK クライアントを計測するためのサポートを説明するために、アプリケーションはトレースハンドラーをユーザー、ゲーム、セッションモデルのリクエストハンドラーAmazonDynamoDBClientBuilderとして に渡します。このコード変更により、これらのクライアントを使用する DynamoDB に対するすべての呼び出しを実装するように SDK に指示します。

Example **src/main/java/scorekeep/SessionModel.java** – AWS SDK クライアントの手動実装

```
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);
```

プロジェクトの依存関係から AWS SDK Instrumentor サブモジュールを削除すると、手動で計測された AWS SDK クライアントのみがトレースマップに表示されます。

追加のサブセグメントを作成する

ユーザーモデルクラスでは、アプリケーションがサブセグメントを手動で作成して、saveUser 関数内で行われるすべてのダウンストリーム呼び出しをグループ化し、メタデータを追加します。

Example **src/main/java/scorekeep/UserModel.java** - カスタムサブセグメント

```
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();
    }
  }
```

注釈、メタデータ、およびユーザー ID を記録する

ゲームモデルクラスでは、アプリケーションはDynamoDB にゲームを保存するたびにメタデータ ブロックに Game オブジェクトを記録します。アプリケーションは個別にゲーム ID を、 フィルタ式 で使用できるように、注釈 に記録します。

Example **src/main/java/scorekeep/GameModel.java**–注釈とメタデータ

```
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();
    }
  }
```

移動コントローラーでは、アプリケーションはユーザー ID を setUser を使用して記録します。ユーザー ID はセグメントの個別のフィールドに記録され、検索用にインデックスが作成されます。

Example src/main/java/scorekeep/MoveController.java – ユーザー ID

```
import com.amazonaws.xray.AWSXRay;
...
  @RequestMapping(value="/{userId}", method=RequestMethod.POST)
  public Move newMove(@PathVariable String sessionId, @PathVariable String
 gameId, @PathVariable String userId, @RequestBody String move) throws
 SessionNotFoundException, GameNotFoundException, StateNotFoundException,
 RulesException {
    AWSXRay.getCurrentSegment().setUser(userId);
    return moveFactory.newMove(sessionId, gameId, userId, move);
  }
```

送信 HTTP 呼び出しの計測

ユーザーファクトリクラスは、アプリケーションがX-Ray SDK for Javaの HTTPClientBuilder バージョンを使用して送信 HTTP 呼び出しを計測する方法を示します。

Example **src/main/java/scorekeep/UserFactory.java**–HTTP クライアント計測

```
import com.amazonaws.xray.proxies.apache.http.HttpClientBuilder;

  public String randomName() throws IOException {
    CloseableHttpClient httpclient = HttpClientBuilder.create().build();
    HttpGet httpGet = new HttpGet("http://uinames.com/api/");
    CloseableHttpResponse response = httpclient.execute(httpGet);
    try {
      HttpEntity entity = response.getEntity();
```

```
      InputStream inputStream = entity.getContent();
      ObjectMapper mapper = new ObjectMapper();
      Map<String, String> jsonMap = mapper.readValue(inputStream, Map.class);
      String name = jsonMap.get("name");
      EntityUtils.consume(entity);
      return name;
    } finally {
      response.close();
    }
  }
```

現在 `org.apache.http.impl.client.HttpClientBuilder` を使用している場合は、そのクラスのインポートステートメントを `com.amazonaws.xray.proxies.apache.http.HttpClientBuilder` のものと交換するだけです。

PostgreSQL データベースに対する呼び出しの計測

`application-pgsql.properties` ファイルは X-Ray PostgreSQL トレースインターセプターを `RdsWebConfig.java` で作成されたデータソースに追加します。

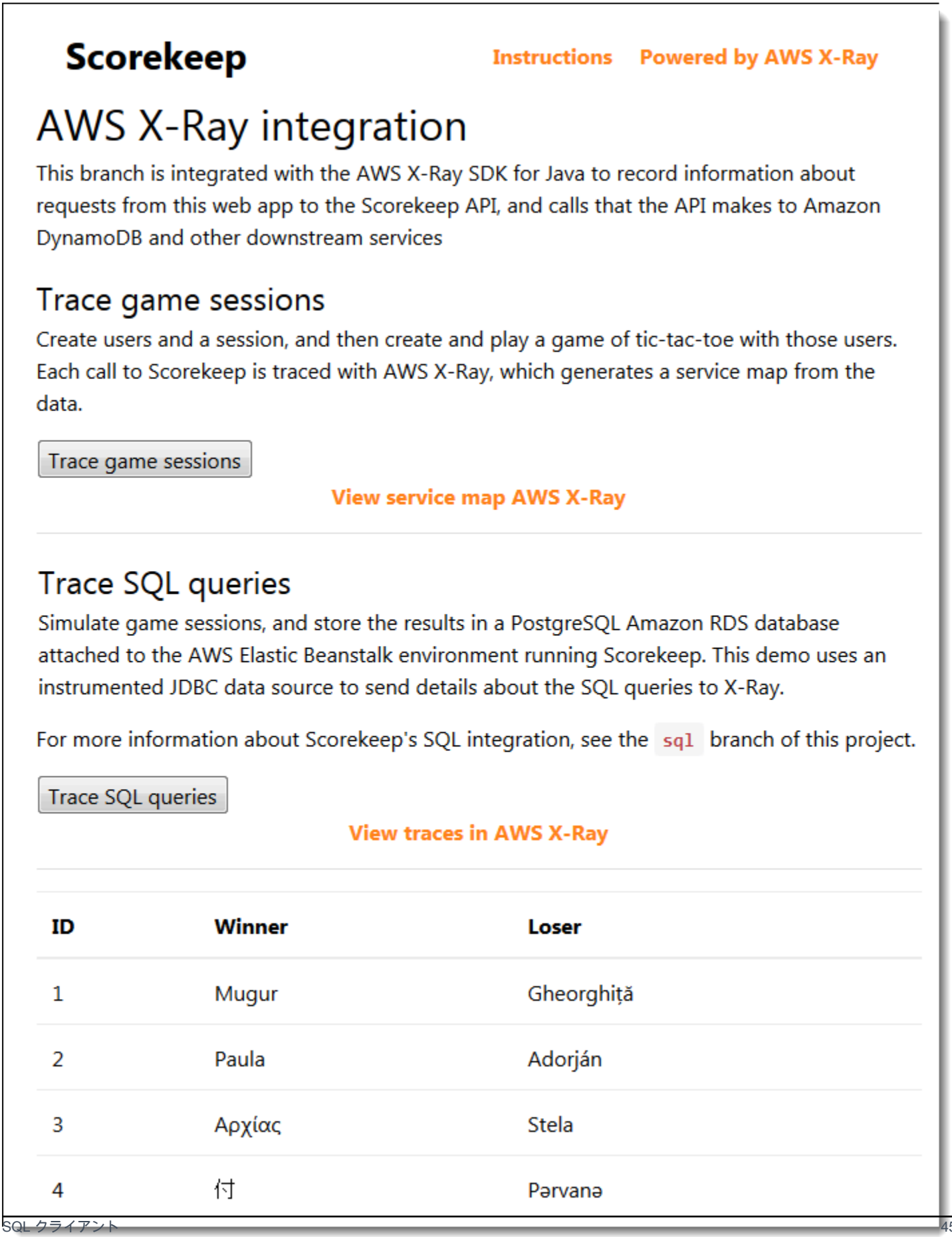
Example **`application-pgsql.properties`** – PostgreSQL データベース実装

```
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**
>
> アプリケーション環境に PostgreSQL データベースを追加する方法に関する詳細については、『https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.managing.db.html 開発者ガイド』の「AWS Elastic BeanstalkElastic Beanstalk を使用してデータベースを設定する」 を参照してください。

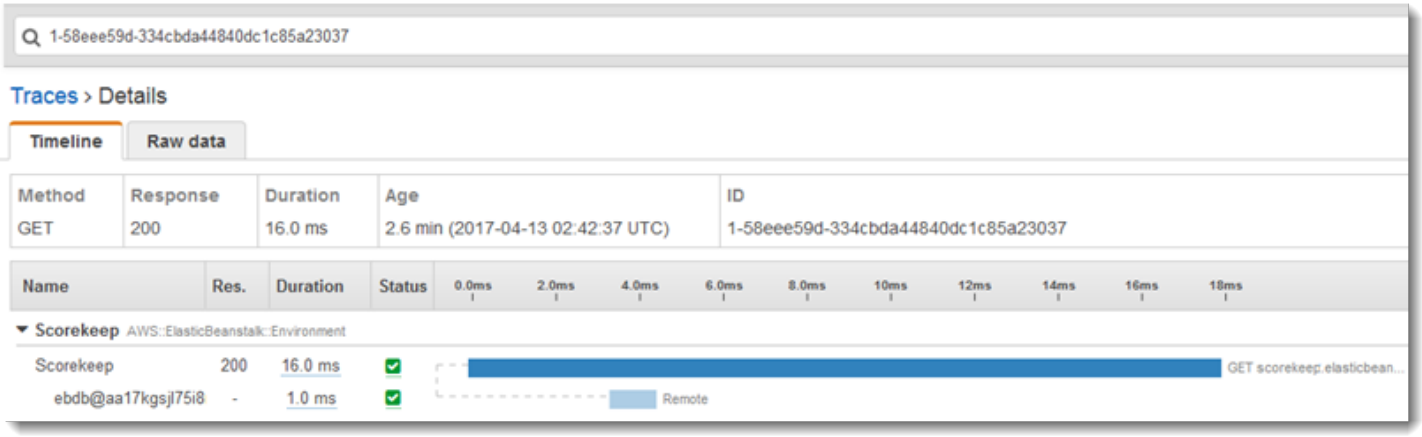
`xray` ブランチの X-Rayデモページは、実装済みデータソースを使用して、生成した SQL クエリに関する情報を示すトレースを生成するデモを含みます。実行中のアプリケーションの `/#/xray` パス

に移動するか、ナビゲーションバーの [Powered by AWS X-Ray] を選択してデモページを表示します。


Scorekeep
Instructions
Powered by AWS X-Ray
AWS X-Ray integration
This branch is integrated with the AWS X-Ray SDK for Java to record information about requests from this web app to the Scorekeep API, and calls that the API makes to Amazon DynamoDB and other downstream services
Trace game sessions
Create users and a session, and then create and play a game of tic-tac-toe with those users. Each call to Scorekeep is traced with AWS X-Ray, which generates a service map from the data.
Trace game sessions
View service map AWS X-Ray
Trace SQL queries
Simulate game sessions, and store the results in a PostgreSQL Amazon RDS database attached to the AWS Elastic Beanstalk environment running Scorekeep. This demo uses an instrumented JDBC data source to send details about the SQL queries to X-Ray.
For more information about Scorekeep's SQL integration, see the sql branch of this project.
Trace SQL queries
View traces in AWS X-Ray
ID
Winner
Loser
1
Mugur
Gheorghiță
2
Paula
Adorján
3
Αρχίας
Stela
4
付
Pərvanə

[Trace SQL queries] を選択して、ゲームセッションをシミュレートし、アタッチされたデータベースに結果を保存します。次に、[View traces in AWS X-Ray (X-Ray でトレースを表示)] を選択して、API の `/api/history` ルートにヒットするトレースのフィルタリングされたリストを表示します。

SQL クエリを含め、リストからトレースのいずれかを選択して、タイムラインを表示します。



AWS Lambda 関数の計測

Scorekeep は 2 つの AWS Lambda 関数を使用します。1 つ目は、新しいユーザー用にランダムな名前を生成する `lambda` ブランチからの Node.js 関数です。ユーザーが名前を入力せずにセッションを作成すると、アプリケーションは、 AWS SDK for Javaで `random-name` という名前の関数を呼び出します。X-Ray SDK for Java は、計測された AWS SDK クライアントで行われた他の呼び出しと同様に、Lambda への呼び出しに関する情報をサブセグメントに記録します。

> **Note**
>
> `random-name` Lambda 関数を実装するには、Elastic Beanstalk 環境の外でその他のリソースを作成する必要があります。詳細については readme、手順については「AWS Lambda との統合」を参照してください。

2 つ目の関数 `scorekeep-worker` は、Scorekeep API と関係なく実行される Python 関数です。ゲームが終了すると、API はセッション ID とゲーム ID を SQS キューに書き込みます。ワーカー関数はキューから項目を読み取り、Scorekeep API を呼び出してAmazon S3 内のストレージのゲームセッションごとに完全なレコードを構築します。

Scorekeep には、両方の関数を作成するための AWS CloudFormation テンプレートとスクリプトが含まれています。X-Ray SDK を関数コードとバンドルする必要があるため、テンプレートはコードなしで関数を作成します。Scorekeep をデプロイすると、`.ebextensions` フォルダに含まれている設定ファイルにより、SDK を含むソースバンドルが作成され、 AWS Command Line Interfaceによって関数コードと設定が更新されます。

関数

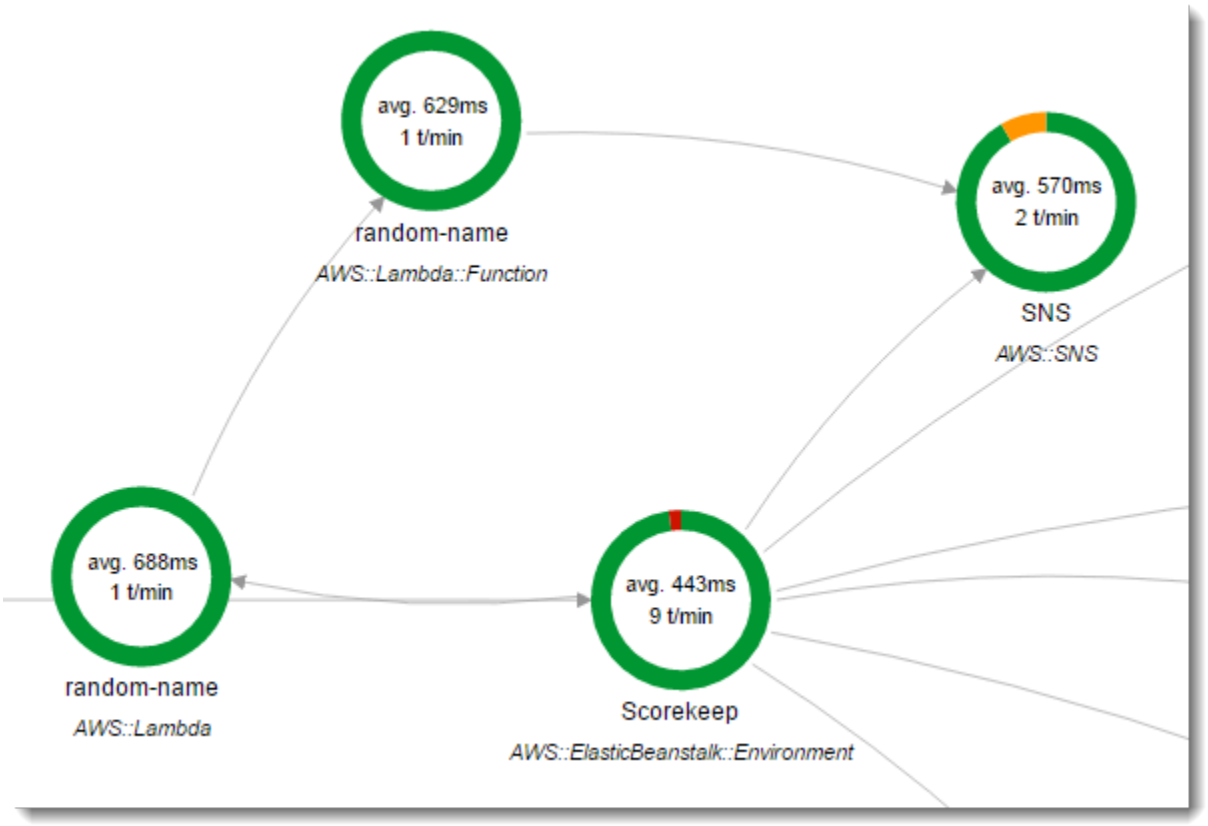
- ランダム名
- ワーカー

ランダム名

Scorekeep は、ユーザーがサインインしたりユーザー名を指定したりせずにゲームセッションを開始するとランダム名関数を呼び出します。`random-name` の呼び出しが Lambda で処理されると、「トレースヘッダー」が読み出されます。これには、X-Ray SDK for Javaによって書き込まれるトレース ID とサンプリングデシジョンを含みます。

Lambda は、サンプリングされたリクエストごとに X-Ray デーモンを実行し、2 つのセグメントを書き込みます。最初のセグメントでは、関数を呼び出す Lambda の呼び出しに関する情報を記録します。このセグメントには、Scorekeep によって記録されるサブセグメントと同じ情報が含まれますが、Lambda の視点からという点で異なります。2 番目のセグメントは、関数の動作を表します。

Lambda は、関数コンテキストを通じて X-Ray SDK に関数セグメントを渡します。Lambda 関数を実装した場合、受信リクエストのセグメントを作成するために SDK は使用しません。 Lambdaにセグメントが提供され、SDK を使用することでクライアントを実装してサブセグメントを書き込みます。



random-name 関数は、Node.js で実装されています。Node.js JavaScript の SDK を使用して Amazon SNS で通知を送信し、X-Ray SDK for Node.js を使用して AWS SDK クライアントを計測します。注釈を書き込むため、関数は AWSXRay.captureFunc を使用してカスタムサブセグメントを作成し、実装された関数に注釈を書き込みます。 Lambdaでは、関数セグメントに直接注釈を書き込むことはできません。作成したサブセグメントにのみ書き込むことができます。

Example **function/index.js** - Random 名 Lambda 関数

```
var AWSXRay = require('aws-xray-sdk-core');
var AWS = AWSXRay.captureAWS(require('aws-sdk'));

AWS.config.update({region: process.env.AWS_REGION});
var Chance = require('chance');

var myFunction = function(event, context, callback) {
  var sns = new AWS.SNS();
  var chance = new Chance();
  var userid = event.userid;
  var name = chance.first();

  AWSXRay.captureFunc('annotations', function(subsegment){
```

```
    subsegment.addAnnotation('Name', name);
    subsegment.addAnnotation('UserID', event.userid);
  });

  // Notify
  var params = {
    Message: 'Created randon name "' + name + '"" for user "' + userid + '".',
    Subject: 'New user: ' + name,
    TopicArn: process.env.TOPIC_ARN
  };
  sns.publish(params, function(err, data) {
    if (err) {
      console.log(err, err.stack);
      callback(err);
    }
    else {
      console.log(data);
      callback(null, {"name": name});
    }
  });
};

exports.handler = myFunction;
```

この関数は、サンプルアプリケーションを Elastic Beanstalk にデプロイするときに自動的に作成されます。xray ブランチには、空白のLambda 関数を作成するスクリプトが含まれています。.ebextensions フォルダ内の設定ファイルは、デプロイnpm install中に で関数パッケージを構築し、 AWS CLI で Lambda 関数を更新します。

ワーカー

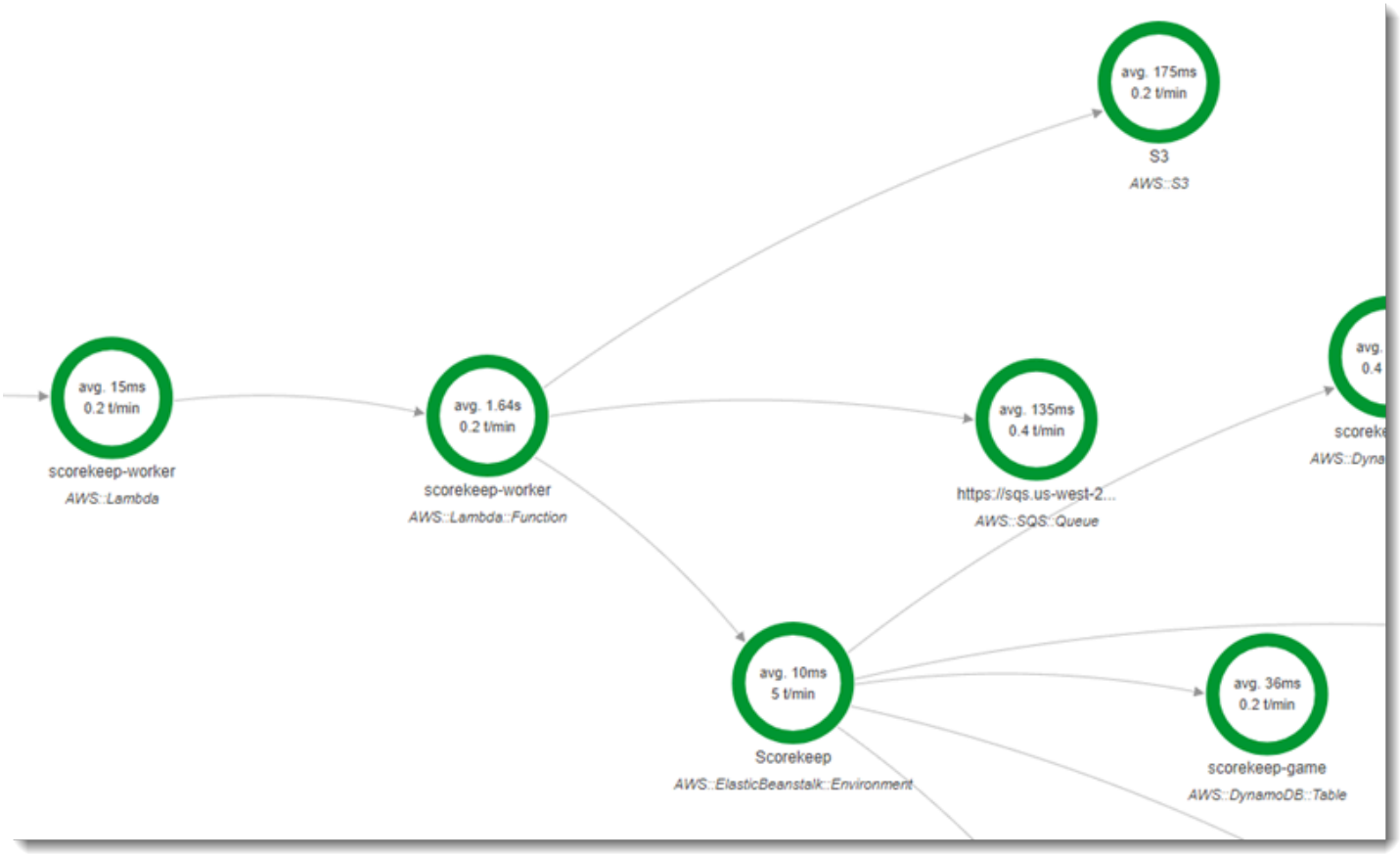
実装されたワーカー関数は、ワーカー関数と関連リソースを作成してからでないと実行できないため、自身のブランチ xray-worker で提供されます。手順については、ブランチの readme を参照してください。

関数は、5 分ごとにバンドルされた Amazon CloudWatch Events イベントによってトリガーされます。実行されると、この関数は Scorekeep によって管理されるAmazon SQS キューから項目を取得します。各メッセージには、完了したゲームに関する情報が含まれています。

ワーカーは、ゲームレコードが参照する他のテーブルからゲームレコードとドキュメントを取得します。たとえば、 DynamoDBのゲームレコードには、ゲーム中に実行されたムービーのリストが含ま

れています。リストには、ムービー自体は含まれておらず、別個のテーブルに格納されたムービーの ID が含まれています。

セッションと状態は、参照としても保存されます。これにより、ゲームテーブル内のエントリが大きくなりすぎることはありませんが、ゲームに関するすべての情報を取得するには追加の呼び出しが必要です。このワーカーは、これらのすべてのエントリを間接参照し、ゲームの完全なレコードをAmazon S3 に単一のドキュメントとして構築します。データで分析を行う場合、読み取り量の多いデータ移行を実行してデータを DynamoDBから取得しなくても、Amazon Athena を使用して Amazon S3 で直接クエリを実行できます。



ワーカー関数では、 AWS Lambdaのその設定においてアクティブトレースが有効になっています。ランダム名関数とは異なり、ワーカーは計測されたアプリケーションからリクエストを受信しないため、 AWS Lambda はトレースヘッダーを受信しません。アクティブトレースを使用すると、Lambdaはトレース ID を作成してサンプリングの決定を行います。

X-Ray SDK for Python は、 SDK をインポートし、そのpatch_all関数を実行して、Amazon SQS AWS SDK for Python (Boto) と Amazon S3 の呼び出しに使用する と HTTclients にパッチを適用する関数の上部にある数行にすぎません。 Amazon SQS ワーカーが API を呼び出すと、SDK はトレースヘッダーをリクエストに追加し、API を通じて呼び出すをトレースします。

Example _lambda/scorekeep-worker/scorekeep-worker.py -- ワーカー Lambda 関数

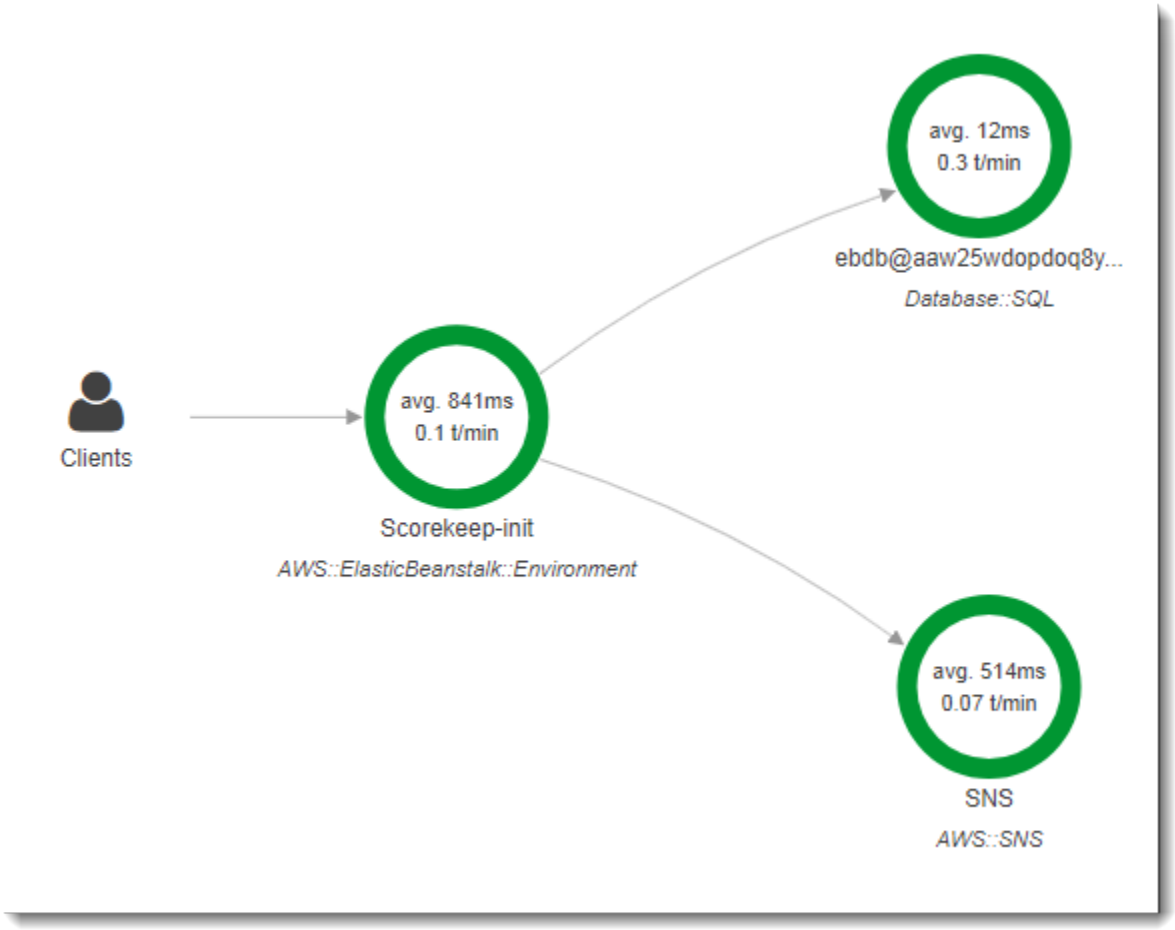
```
import os
import boto3
import json
import requests
import time
from aws_xray_sdk.core import xray_recorder
from aws_xray_sdk.core import patch_all

patch_all()
queue_url = os.environ['WORKER_QUEUE']

def lambda_handler(event, context):
    # Create SQS client
    sqs = boto3.client('sqs')
    s3client = boto3.client('s3')

    # Receive message from SQS queue
    response = sqs.receive_message(
        QueueUrl=queue_url,
        AttributeNames=[
            'SentTimestamp'
        ],
        MaxNumberOfMessages=1,
        MessageAttributeNames=[
            'All'
        ],
        VisibilityTimeout=0,
        WaitTimeSeconds=0
    )
   ...
```

スタートアップコードの作成

X-Ray SDK for Java は、着信リクエストのセグメントを自動的に作成します。リクエストが範囲内にある限り、計測クライアントを使用して問題なしでサブセグメントを記録できます。しかし、計測クライアントをスタートアップコードで使用しようとすると、SegmentNotFoundException が発生します。

スタートアップコードは、ウェブアプリケーションの標準的なリクエスト/レスポンスフローの外側で実行されるため、手動でセグメントを作成して計測する必要があります。Scorekeep はスタート

アップコードのインストルメンテーションをWebConfig ファイルに表示します。Scorekeep はスタートアップ時に SQL データベースと Amazon SNS を呼び出します。



デフォルトの WebConfig クラスは、通知のための Amazon SNS サブスクリプションを作成します。Amazon SNSクライアントの使用時に X-Ray SDK が書き込むセグメントを提供するために、Scorekeep はグローバルレコーダー上で beginSegment と endSegment を呼び出します。

Example **src/main/java/scorekeep/WebConfig.java** – スタートアップコードの計測 AWS SDK クライアント

```
AWSXRay.beginSegment("Scorekeep-init");
if ( System.getenv("NOTIFICATION_EMAIL") != null ){
  try { Sns.createSubscription(); }
  catch (Exception e ) {
    logger.warn("Failed to create subscription for email "+
 System.getenv("NOTIFICATION_EMAIL"));
  }
}
```

```
AWSXRay.endSegment();
```

Amazon RDS データベースが接続されているときに Scorekeep が使用する `RdsWebConfig` では、スタートアップ時にデータベーススキーマを適用するときに Hibernate が使用する SQL クライアントのセグメントも作成されます。

Example **src/main/java/scorekeep/RdsWebConfig.java** – スタートアップコードの実装 SQL データベースクライアント

```
@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` は自動的に実行され、SQL クライアントを使用します。クライアントが計測されているため、Scorekeep はデフォルトの実装をオーバーライドし、SDK がクライアントの呼び出し時に使用するセグメントを提供する必要があります。

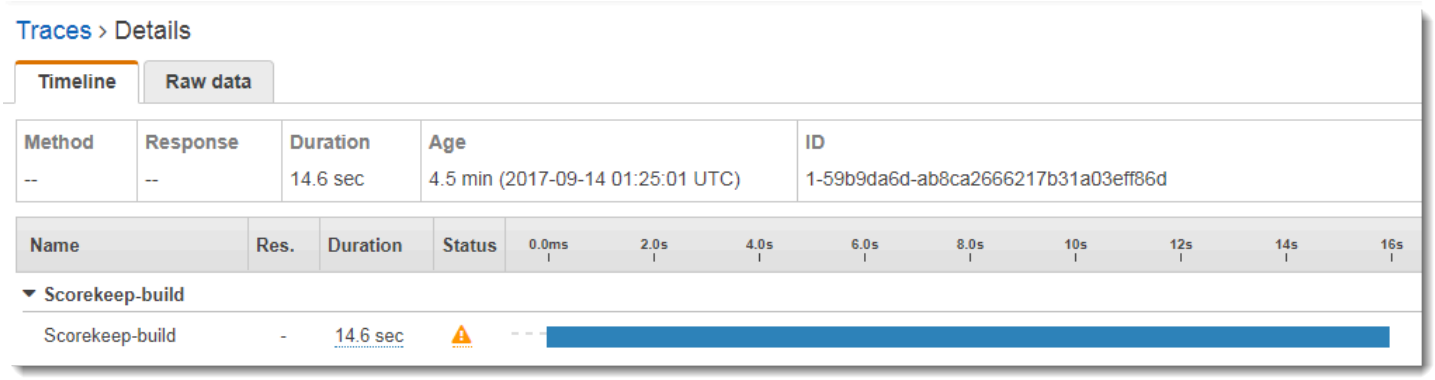
実装スクリプト

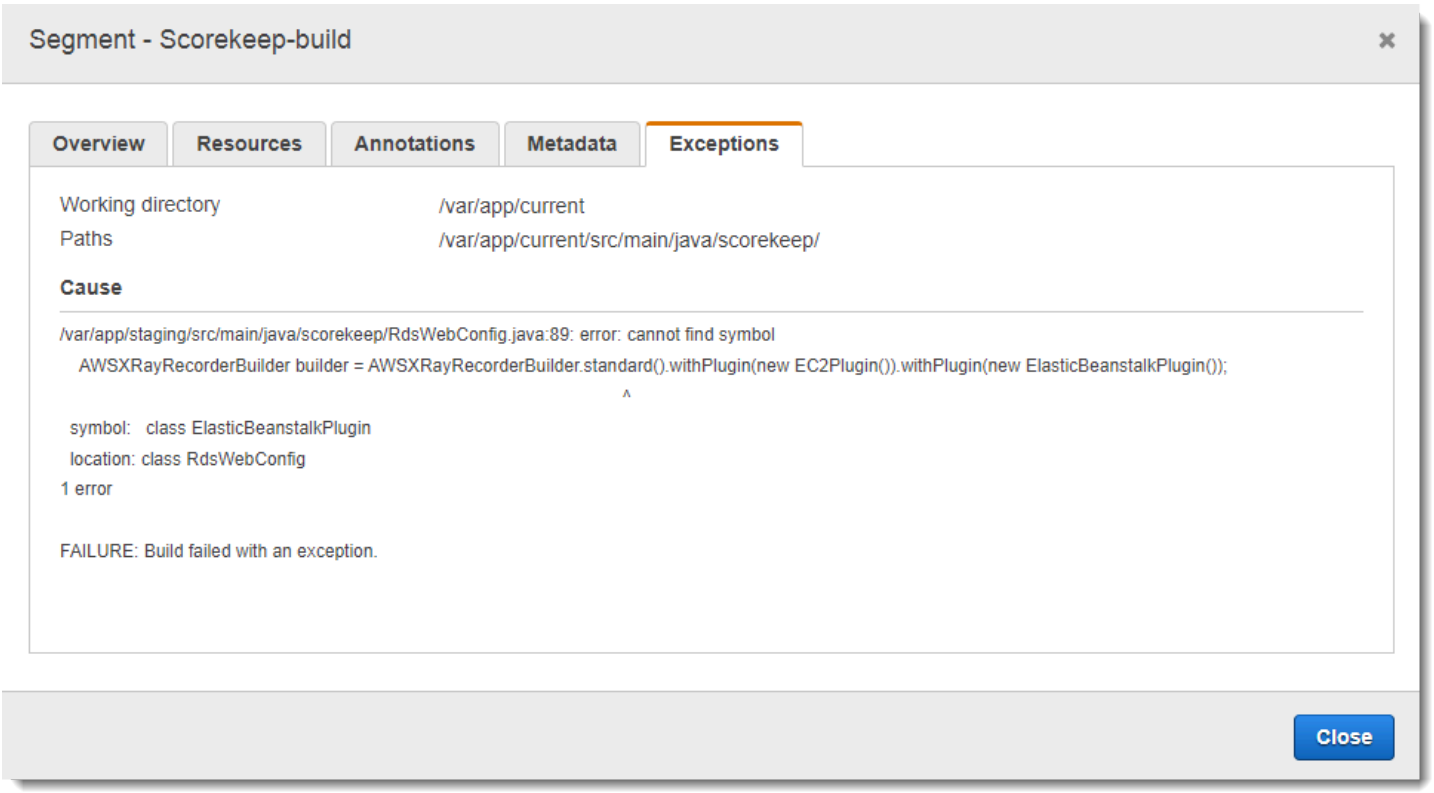
また、アプリケーションの一部ではないコードを計測することもできます。X-Ray デーモンが実行されている場合、X-Ray SDK によって生成されない場合でも、デーモンは受信したすべてのセグメントを X-Ray に中継します。Scorekeep は独自のスクリプトを使用して、展開中にアプリケーションをコンパイルするビルドを実装します。

Example **bin/build.sh** – 実装されたビルドスクリプト

```
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、および xray_success.py は、セグメントオブジェクトを構築し、JSON 文書に変換し、UDP 経由でデーモンに送信する単純な Python スクリプトです。Gradle ビルドが失敗した場合、X-Ray コンソールトレースマップの Scorekeep-build ノードをクリックすると、エラーメッセージを見つけることができます。





ウェブアプリケーションクライアントの実装

xray-cognito ブランチで、Scorekeep は Amazon Cognito を使用して、ユーザーがアカウントを作成し、それを使用してサインインして Amazon Cognito ユーザープールからユーザー情報を取得することができるようにします。ユーザーがサインインすると、Scorekeep は Amazon Cognito ID プールを使用して、 で使用する一時的な AWS 認証情報を取得します AWS SDK for JavaScript。

ID プールは、サインインしたユーザーがトレースデータを AWS X-Rayに書き込むことができるように設定されています。ウェブアプリケーションは、これらの認証情報を使用して、ログインしたユーザーの ID、ブラウザパス、および Scorekeep API への呼び出しのクライアントビューを記録します。

ほとんどの作業は `xray` というサービスクラスで行われます。このサービスクラスは、必要な識別子の生成、進行中のセグメントの作成、セグメントのファイナライズ、セグメントドキュメントの X-Ray API への送信手段を提供します。

Example **public/xray.js** – セグメントの記録とアップロード

```
...
  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);
      }
```

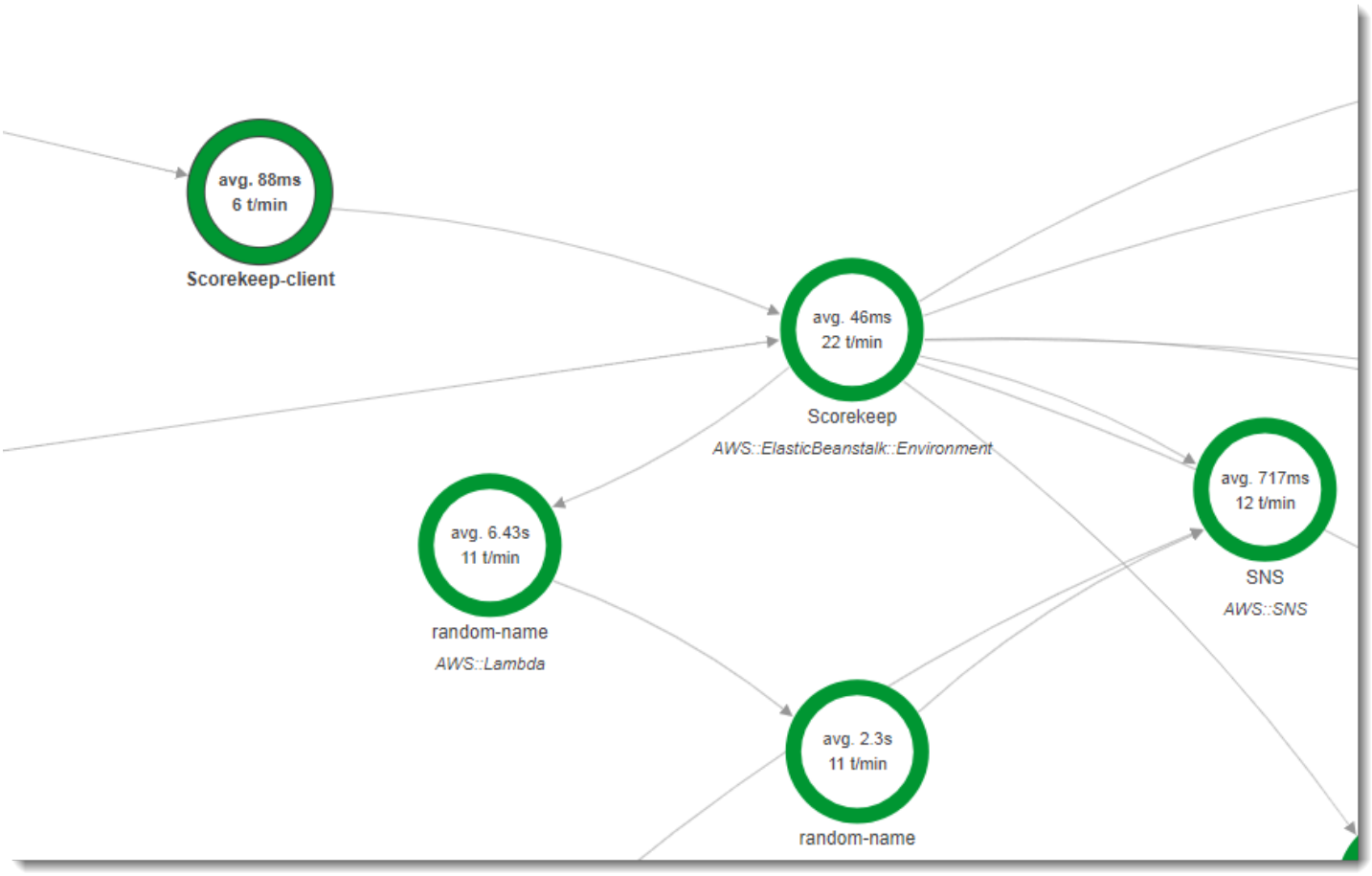
```
    })
  }
```

これらのメソッドは、ウェブアプリケーションが Scorekeep API を呼び出すために使用するリソースサービスのヘッダーおよび `transformResponse` 関数で呼び出されます。API が生成するセグメントと同じトレースにクライアントセグメントを含めるには、ウェブアプリケーションは、X-Ray SDK が読み取り可能なトレースヘッダー (`X-Amzn-Trace-Id`) にトレース ID とセグメント ID を含める必要があります。実装された Java アプリケーションがこのヘッダーでリクエストを受け取ると、X-Ray SDK for Java は同じトレース ID を使用し、ウェブアプリケーションクライアントからのセグメントをセグメントの親にします。

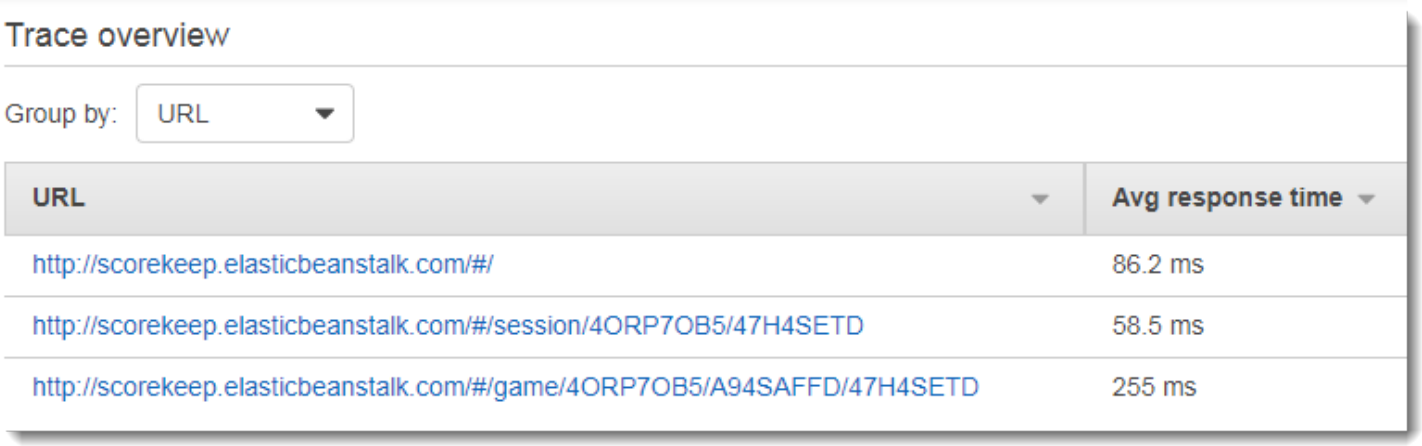
Example **`public/app/services.js`** – 角度リソースコールとトレースヘッダーの書き込みセグメントの記録

```
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);
      },
    },
...
```

結果のトレースマップには、ウェブアプリクライアントのノードが含まれます。



ウェブアプリケーションからのセグメントを含むトレースには、ユーザーがブラウザに表示する URL (/#/ で始まるパス) が表示されます。クライアント実装機能がなければ、ウェブアプリケーションが呼び出す API リソース (/api/ で始まるパス) の URL のみを取得します。



実装されたクライアントをワーカースレッドで使用する

Scorekeep は、ユーザーがゲームに勝利したときにワーカースレッドを使用して Amazon SNS に通知を発行します。通知の発行は、残りのリクエスト操作の組み合わせよりも時間がかかり、クライアントまたはユーザーには影響しません。したがって、タスクを非同期で実行することは、応答時間を改善するための良い方法です。

ただし、X-Ray SDK for Java は、スレッドが作成されたときにどのセグメントがアクティブであったかを認識しません。結果として、実装された AWS SDK for Java クライアントをスレッド内で使用しようとすると、`SegmentNotFoundException` がスローされ、スレッドがクラッシュします。

Example web-1.error.log

```
Exception in thread "Thread-2" com.amazonaws.xray.exceptions.SegmentNotFoundException:
 Failed to begin subsegment named 'AmazonSNS': segment cannot be found.
        at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
        at
 sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:62)
        at
 sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.ja
...
```

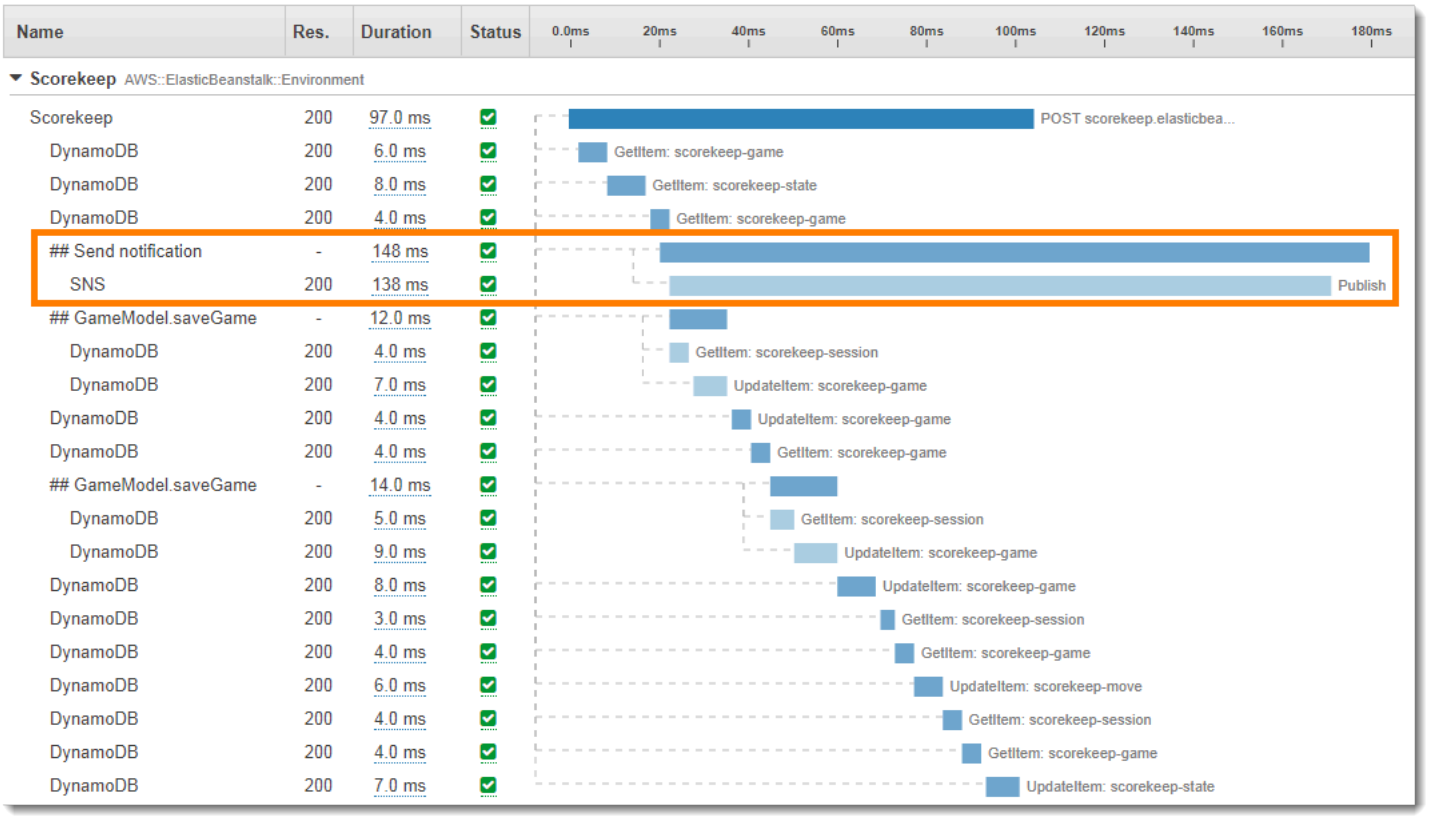
この問題を解決するために、アプリケーションは `GetTraceEntity` を使用してメインスレッドのセグメントへの参照を取得し、`Entity.run()` を使用してセグメントのコンテキストにアクセスし、ワーカースレッドコードを安全に実行します。

Example **src/main/java/scorekeep/MoveFactory.java** – ワーカースレッドにトレースコンテキストを渡す

```
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();
        }
    }
```

Amazon SNS の呼び出しの前に要求が解決されるため、アプリケーションはスレッド用に別のサブセグメントを作成します。これにより、Amazon SNS からの応答を記録する前に X-Ray SDK がセグメントを閉じるのを防ぎます。Scorekeep がリクエストを解決したときにサブセグメントが開いていない場合、Amazon SNS からの応答は失われる可能性があります。



マルチスレッドの詳細については、「マルチスレッドアプリケーションでのスレッド間のセグメントコンテキストの受け渡し」を参照してください。

トラブルシューティング AWS X-Ray

このトピックでは、X-Ray、API、コンソール、または SDK を使用する際に発生する可能性のある一般的なエラーと問題を示します。ここに記載されていない問題が見つかった場合は、このページの [Feedback] ボタンを使用して報告することができます。

セクション

X-Ray トレースマップとトレースの詳細ページ

以下のセクションは、X-Ray トレースマップとトレースの詳細ページを使用して問題が発生した場合に役立ちます。

すべての CloudWatch ログが表示されない

X-Ray トレースマップとトレースの詳細ページに表示されるようにログを設定する方法は、サービスによって異なります。

- API Gateway でログが有効になっている場合は、API Gateway のログが表示されます。

すべてのサービスマップノードが、関連するログの表示をサポートしているわけではありません。次のノードタイプのログを表示します。

- Lambda コンテキスト
- Lambda 関数
- API Gateway ステージ
- Amazon ECS クラスター
- Amazon ECS インスタンス
- Amazon ECS サービス
- Amazon ECS タスク

- Amazon EKS クラスター
- Amazon EKS 名前空間
- Amazon EKS ノード
- Amazon EKS ポッド
- Amazon EKS サービス

X-Ray トレースマップにすべてのアラームが表示されない

X-Ray トレースマップには、ノードに関連付けられているアラームが ALARM 状態にある場合にのみ、ノードのアラートアイコンが表示されます。

トレースマップは、次のロジックを使用してアラームをノードに関連付けます。

- ノードが AWS サービスを表す場合、そのサービスに関連付けられた 名前空間を持つすべてのアラームがノードに関連付けられます。例えば、タイプ のノード`AWS::Kinesis`は、 CloudWatch 名前空間 のメトリクスに基づくすべてのアラームにリンクされます`AWS/Kinesis`。
- ノードが AWS リソースを表す場合、その特定のリソースのアラームがリンクされます。例えば、`MyTable``AWS::DynamoDB::Table`「」という名前のタイプ のノードは、 名前空間のメトリクスに基づいており、`AWS/DynamoDB``TableName`ディメンションが に設定されているすべてのアラームにリンクされます`MyTable`。
- ノードのタイプが不明で、名前の周囲に破線で囲まれている場合、そのノードにはアラームが関連付けられていません。

トレースマップに一部の AWS リソースが表示されない

すべての AWS リソースが専用ノードで表されるわけではありません。一部の AWS サービスは、サービスへのすべてのリクエストに対して単一のノードで表されます。次のリソースタイプは、リソースごとのノードとともに表示されます。

- `AWS::DynamoDB::Table`
- `AWS::Lambda::Function`

  Lambda 関数は 2 つのノードで表されます。1 つは Lambda コンテナ用、もう 1 つは関数用です。これは、Lambda 関数のコールドスタートの問題を特定するのに役立ちます。Lambda コンテナノードは、Lambda 関数ノードと同じ方法でアラームとダッシュボードに関連付けられます。
- `AWS::ApiGateway::Stage`

- `AWS::SQS::Queue`
- `AWS::SNS::Topic`

トレースマップにノードが多すぎる

X-Ray グループを使用して、マップを複数のマップに分割します。詳細については、「グループでフィルター式を使用する」を参照してください。

X-Ray SDK for Java

エラー： Exception in thread "Thread-1" com.amazonaws.xray.exceptions.SegmentNotFoundException: Failed to begin subsegment named 'AmazonSNS ': segment cannot found.

このエラーは、X-Ray SDK が への発信呼び出しを記録しようとしたが AWS、オープンセグメントを見つけられなかったことを示します。これは次の場合に発生する可能性があります。

- サーブレットフィルタが設定されていない X–Ray、 SDK は、 という名前のフィルターを使用して、着信リクエストのセグメントを作成しますAWSXRayServletFilter。着信リクエストを計測するためのサーブレットフィルターを設定します。
- サーブレットコード以外の計測クライアントを使用しています 計測クライアントを使用して、スタートアップコードまたは着信リクエストに応答して実行されないその他のコードで呼び出しを行う場合は、セグメントを手動で作成する必要があります。例については、「スタートアップコードの作成」を参照してください。
- ワーカースレッドで計測クライアントを使用しています 新しいスレッドを作成すると、X-Ray レコーダーはオープンセグメントへの参照を失います。getTraceEntity および setTraceEntityメソッドを使用して、現在のセグメントまたはサブセグメント (Entity) への参照を取得し、スレッド内のレコーダーに戻すことができます。例については、「実装されたクライアントをワーカースレッドで使用する」を参照してください。

Node.jsに使われる X-Ray SDK

問題: Sequelize で CLS が機能しない

clsの方法で Node.js の名前空間で使われるX-Ray SDKをSequelizeへ渡します。

```
var AWSXRay = require('aws-xray-sdk');
```

```
const Sequelize = require('sequelize');
Sequelize.cls = AWSXRay.getNamespace();
const sequelize = new Sequelize(...);
```

問題: Bluebird で CLS が機能しない

`cls-bluebird` を使用して、Bluebird が CLS で動作するようにします。

```
var AWSXRay = require('aws-xray-sdk');
var Promise = require('bluebird');
var clsBluebird = require('cls-bluebird');
clsBluebird(AWSXRay.getNamespace());
```

X-Ray デーモン

問題: デーモンが間違った認証情報を使用する

デーモンは AWS SDK を使用して認証情報をロードします。認証情報を提供する複数の方法を使用する場合は、優先順位が最も高い方法が使用されます。詳細については、「デーモンを実行する」を参照してください。

のセキュリティ AWS X-Ray

のクラウドセキュリティが最優先事項 AWS です。お客様は AWS 、セキュリティを最も重視する組織の要件を満たすように構築されたデータセンターとネットワークアーキテクチャからメリットを得られます。

セキュリティは、 AWS とユーザーの間で共有される責任です。責任共有モデルでは、これをクラウドのセキュリティおよびクラウド内のセキュリティとして説明しています。

- クラウドのセキュリティ — AWS は、 AWS のサービス で実行されるインフラストラクチャを保護する責任を担います AWS クラウド。また、 は、安全に使用できるサービス AWS も提供します。セキュリティの有効性は、AWS コンプライアンスプログラムの一環として、サードパーティーの審査機関によって定期的にテストおよび検証されています。X-Ray に適用されるコンプライアンスプログラムの詳細については、「コンプライアンスプログラムの対象範囲となるAWS のサービス」を参照してください。
- クラウドのセキュリティ — お客様の責任は AWS のサービス 、使用する によって決まります。また、お客様は、お客様のデータの機密性、組織の要件、および適用可能な法律および規制などの他の要因についても責任を担います。

このドキュメントは、X-Ray を使用するときに、共有責任モデルを適用する方法を理解するのに役立ちます。以下のトピックでは、セキュリティおよびコンプライアンスの目的を達成するために X-Ray を設定する方法を示します。また、X-Ray リソースのモニタリングや保護 AWS のサービス に役立つ他の の使用方法についても説明します。

トピック

AWS X-Ray でのデータ保護

AWS X-Ray は常に保管時のトレースと関連データを暗号化します。コンプライアンスの要件や内部的な要件に応じて暗号化キーを監査して無効にする必要がある場合は、X-Ray を設定してデータを暗号化するように AWS Key Management Service (AWS KMS) キーを使用することができます。

X-Ray は `aws/xray` という名前の AWS マネージドキー を指定します。AWS CloudTrail でキーの使用状況を監査するだけで、キー自体を管理する必要がない場合は、このキーを使用します。キーへのアクセスを管理したり、キーの更新を設定したりする必要がある場合は、カスタマー管理のキーを作成できます。

暗号化設定を変更すると、X-Ray でのデータキーの生成および伝達に少し時間がかかります。新しいキーの処理中に、X-Ray は新しい設定と古い設定を組み合わせてデータを暗号化することがあります。暗号化設定を変更するときに、既存のデータは再暗号化されません。

> **Note**
>
> X-Ray が KKMS キーを使用してトレースデータを暗号化または復号するときに、AWS KMS の料金が発生します。
>
> - デフォルトの暗号化 – 無料。
> - AWS マネージドキー – キーの使用について料金が発生します。
> - カスタマー管理キー – キーの保存と使用について料金が発生します。
>
> 詳細については、「AWS Key Management Service の料金」を参照してください。

> **Note**
>
> X-Ray インサイト通知は現在カスタマー管理キーをサポートしていない Amazon EventBridge にイベントを送信します。詳細については、「Amazon EventBridge におけるデータ保護』を参照してください。

カスタマー管理キーを使用して暗号化されたトレースを表示するように X-Ray を設定するには、カスタマー管理キーに対するユーザーレベルのアクセス権が必要です。詳細については、「暗号化のユーザーアクセス許可」を参照してください。

CloudWatch console

CloudWatch コンソールを使用して暗号化に KMS キーを使用するように X-Ray を設定するには

1. AWS Management Console にサインインして、CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. 左側のナビゲーションペインの [Settings] (設定) を選択します。
3. X-Ray トレースセクションの [暗号化] の下にある [設定を表示] を選択します。
4. [暗号化の設定] セクションで、[編集] を選択します。
5. [KMS キーを使用] を選択します。
6. ドロップダウンメニューからキーを選択します。
   - aws/xray – AWS マネージドキー を使用します。
   - キーエイリアス – アカウントでカスタマー管理キーを使用します。
   - キー ARN を手動で入力 – 別のアカウントのカスタマー管理キーを使用します。表示されるフィールドに、キーの完全な Amazon リソースネーム (ARN) を入力します。
7. [暗号化の更新] を選択します。

X-Ray console

X-Ray コンソールを使用して暗号化に KMS キーを使用するように X-Ray を設定するには

1. [X-Ray console (X-Ray コンソール)] を開きます。
2. [暗号化] を選択します。
3. [Use a KMS key (KMS キーを使用する)] を選択します。
4. ドロップダウンメニューからキーを選択します。
   - aws/xray – AWS マネージドキー を使用します。
   - キーエイリアス – アカウントでカスタマー管理キーを使用します。
   - キー ARN を手動で入力 – 別のアカウントのカスタマー管理キーを使用します。表示されるフィールドに、キーの完全な Amazon リソースネーム (ARN) を入力します。
5. [Apply (適用)] を選択します。

> **Note**
>
> X-Ray は非対称 KMS キー をサポートしていません。

X-Ray が暗号化キーにアクセスできない場合、データの保存を停止します。これは、ユーザーが KMS キーにアクセスできなくなった場合や、現在使用されているキーを無効にした場合に発生する可能性があります。この場合、X-Ray はナビゲーションバーに通知を表示します。

X-Ray API を使用して暗号化設定を指定する方法については、「X-Ray API を使用したサンプリング、グループ、および暗号化設定の構成」を参照してください。

の Identity and Access Management AWS X-Ray

AWS Identity and Access Management (IAM) は、管理者が AWS リソースへのアクセスを安全に制御 AWS のサービス するのに役立つ です。IAM 管理者は、誰を認証 (サインイン) し、誰に X-Ray リソースの使用を許可する (アクセス許可を持たせる) かを制御します。IAM は、追加料金なしで AWS のサービス 使用できる です。

トピック

- 対象者
- アイデンティティを使用した認証
- ポリシーを使用したアクセスの管理
- が IAM と AWS X-Ray 連携する方法
- AWS X-Ray アイデンティティベースのポリシーの例
- AWS X-Ray アイデンティティとアクセスに関するトラブルシューティング

対象者

AWS Identity and Access Management (IAM) の使用方法は、X-Ray で行う作業によって異なります。

サービスユーザー - X-Ray サービスを使用してジョブを実行する場合は、必要なアクセス許可と認証情報を管理者が用意します。作業を実行するためにさらに多くの X-Ray の特徴を使用するとき、追加の許可が必要になる場合があります。アクセスの管理方法を理解しておくと、管理者に適切な許可

をリクエストするうえで役立ちます。X-Ray の特徴にアクセスできない場合は、「AWS X-Ray アイデンティティとアクセスに関するトラブルシューティング」を参照してください。

サービス管理者 - 社内の X-Ray リソースを担当している場合は、通常、X-Ray へのフルアクセスがあります。サービスのユーザーがどの X-Ray 機能やリソースにアクセスするかを決めるのは管理者の仕事です。その後、IAM 管理者にリクエストを送信して、サービスユーザーの権限を変更する必要があります。このページの情報を点検して、IAM の基本概念を理解してください。貴社が X-Ray で IAM を利用する方法の詳細については、「が IAM と AWS X-Ray 連携する方法」を参照してください。

IAM 管理者 - IAM 管理者は、X-Ray へのアクセスを管理するポリシーの作成方法の詳細について確認する場合があります。IAM で使用できる X-Ray アイデンティティベースのポリシーの例を表示するには、「AWS X-Ray アイデンティティベースのポリシーの例」を参照してください。

アイデンティティを使用した認証

認証は、ID 認証情報 AWS を使用して にサインインする方法です。として、IAM ユーザーとして AWS アカウントのルートユーザー、または IAM ロールを引き受けることによって認証 (にサインイン AWS) される必要があります。

ID ソース (AWS IAM Identity Center) から提供された認証情報を使用して、フェデレーティッド ID AWS として にサインインできます。IAM Identity Center) ユーザー、会社のシングルサインオン認証、Google または Facebook の認証情報は、フェデレーティッド ID の例です。フェデレーションアイデンティティとしてサインインする場合、IAM ロールを使用して、前もって管理者により ID フェデレーションが設定されています。フェデレーションを使用して にアクセスすると、間接的 AWS にロールを引き受けます。

ユーザーのタイプに応じて、 AWS Management Console または AWS アクセスポータルにサインインできます。へのサインインの詳細については AWS、「 AWS サインイン ユーザーガイド」の「 にサインインする方法 AWS アカウント」を参照してください。

AWS プログラムで にアクセスする場合、 は Software Development Kit (SDK) とコマンドラインインターフェイス (CLI) AWS を提供し、 認証情報を使用してリクエストに暗号で署名します。 AWS ツールを使用しない場合は、リクエストを自分で署名する必要があります。推奨される方法を使用してリクエストを自分で署名する方法の詳細については、「IAM ユーザーガイド」のAWS 「 API リクエストの署名」を参照してください。

使用する認証方法を問わず、セキュリティ情報の提供を追加でリクエストされる場合もあります。例えば、 では、多要素認証 (MFA) を使用してアカウントのセキュリティを高めることを AWS 推奨し

ています。詳細については、「AWS IAM Identity Center ユーザーガイド」の「多要素認証 (MFA)」および「IAM ユーザーガイド」の「AWSでの多要素認証 (MFA) の使用」を参照してください。

AWS アカウント ルートユーザー

を作成するときは AWS アカウント、アカウント内のすべての AWS のサービス およびリソースへの完全なアクセス権を持つ 1 つのサインインアイデンティティから始めます。このアイデンティティは AWS アカウント ルートユーザーと呼ばれ、アカウントの作成に使用した E メールアドレスとパスワードでサインインすることでアクセスできます。日常的なタスクには、ルートユーザーを使用しないことを強くお勧めします。ルートユーザーの認証情報を保護し、それらを使用してルートユーザーのみが実行できるタスクを実行してください。ルートユーザーとしてサインインする必要があるタスクの完全なリストについては、「IAM ユーザーガイド」の「ルートユーザー認証情報が必要なタスク」を参照してください。

IAM ユーザーとグループ

IAM ユーザーは、単一のユーザーまたはアプリケーションに対して特定のアクセス許可 AWS アカウント を持つ 内のアイデンティティです。可能であれば、パスワードやアクセスキーなどの長期的な認証情報を保有する IAM ユーザーを作成する代わりに、一時的な認証情報を使用することをお勧めします。ただし、IAM ユーザーでの長期的な認証情報が必要な特定のユースケースがある場合は、アクセスキーをローテーションすることをお勧めします。詳細については、IAM ユーザーガイドの「長期的な認証情報を必要とするユースケースのためにアクセスキーを定期的にローテーションする」を参照してください。

IAM グループは、IAM ユーザーの集団を指定するアイデンティティです。グループとしてサインインすることはできません。グループを使用して、複数のユーザーに対して一度に権限を指定できます。多数のユーザーグループがある場合、グループを使用することで権限の管理が容易になります。例えば、IAMAdmins という名前のグループを設定して、そのグループに IAM リソースを管理する権限を与えることができます。

ユーザーは、ロールとは異なります。ユーザーは 1 人の人または 1 つのアプリケーションに一意に関連付けられますが、ロールはそれを必要とする任意の人が引き受けるようになっています。ユーザーには永続的な長期の認証情報がありますが、ロールでは一時的な認証情報が提供されます。詳細については、「IAM ユーザーガイド」の「IAM ユーザー (ロールではなく) の作成が適している場合」を参照してください。

IAM ロール

IAM ロールは、特定のアクセス許可 AWS アカウント を持つ 内のアイデンティティです。これは IAM ユーザーに似ていますが、特定のユーザーには関連付けられていません。ロールを切り替える

AWS Management Console ことで、 で IAM ロールを一時的に引き受けることができます。ロールを引き受けるには、 または AWS API オペレーションを AWS CLI 呼び出すか、カスタム URL を使用します。ロールを使用する方法の詳細については、「IAM ユーザーガイド」の「IAM ロールの使用」を参照してください。

一時的な認証情報を持った IAM ロールは、以下の状況で役立ちます。

- フェデレーションユーザーアクセス – フェデレーションアイデンティティに権限を割り当てるには、ロールを作成してそのロールの権限を定義します。フェデレーションアイデンティティが認証されると、そのアイデンティティはロールに関連付けられ、ロールで定義されている権限が付与されます。フェデレーションの詳細については、「IAM ユーザーガイド」の「サードパーティーアイデンティティプロバイダー向けロールの作成」を参照してください。IAM アイデンティティセンターを使用する場合、権限セットを設定します。アイデンティティが認証後にアクセスできるものを制御するため、IAM Identity Center は、権限セットを IAM のロールに関連付けます。権限セットの詳細については、「AWS IAM Identity Center ユーザーガイド」の「権限セット」を参照してください。
- 一時的な IAM ユーザー権限 - IAM ユーザーまたはロールは、特定のタスクに対して複数の異なる権限を一時的に IAM ロールで引き受けることができます。
- クロスアカウントアクセス - IAM ロールを使用して、自分のアカウントのリソースにアクセスすることを、別のアカウントの人物 (信頼済みプリンシパル) に許可できます。クロスアカウントアクセス権を付与する主な方法は、ロールを使用することです。ただし、一部の では AWS のサービス、(ロールをプロキシとして使用する代わりに) リソースにポリシーを直接アタッチできます。クロスアカウントアクセスにおけるロールとリソースベースのポリシーの違いについては、「IAM ユーザーガイド」の「IAM ロールとリソースベースのポリシーとの相違点」を参照してください。
- クロスサービスアクセス — 一部の では、他の の機能 AWS のサービス を使用します AWS のサービス。例えば、あるサービスで呼び出しを行うと、通常そのサービスによって Amazon EC2 でアプリケーションが実行されたり、Amazon S3 にオブジェクトが保存されたりします。サービスでは、呼び出し元プリンシパルの権限、サービスロール、またはサービスにリンクされたロールを使用してこれを行う場合があります。
  - 転送アクセスセッション (FAS) – IAM ユーザーまたはロールを使用して でアクションを実行する場合 AWS、プリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがあります。FAS は、を呼び出すプリンシパルのアクセス許可 AWS のサービス を使用し AWS のサービス、ダウンストリームサービスにリクエストを行うリクエストと組み合わせて使用します。FAS リクエストは、他の AWS のサービス またはリソースとのやり取りを完了する必要があるリクエストを

サービスが受信した場合にのみ行われます。この場合、両方のアクションを実行するためのアクセス許可が必要です。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。

- サービスロール - サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除できます。詳細については、「IAM ユーザーガイド」の「AWS のサービスにアクセス許可を委任するロールの作成」を参照してください。
- サービスにリンクされたロール – サービスにリンクされたロールは、 にリンクされたサービスロールの一種です AWS のサービス。サービスは、ユーザーに代わってアクションを実行するロールを引き受けることができます。サービスにリンクされたロールは に表示され AWS アカウント 、サービスによって所有されます。IAM 管理者は、サービスリンクロールの権限を表示できますが、編集することはできません。

- Amazon EC2 で実行されているアプリケーション – IAM ロールを使用して、EC2 インスタンスで実行され、 AWS CLI または AWS API リクエストを作成しているアプリケーションの一時的な認証情報を管理できます。これは、EC2 インスタンス内でのアクセスキーの保存に推奨されます。EC2 インスタンスに AWS ロールを割り当て、そのすべてのアプリケーションで使用できるようにするには、インスタンスにアタッチされたインスタンスプロファイルを作成します。インスタンスプロファイルにはロールが含まれ、EC2 インスタンスで実行されるプログラムは一時的な認証情報を取得できます。詳細については、「IAM ユーザーガイド」の「Amazon EC2 インスタンスで実行されるアプリケーションに IAM ロールを使用して権限を付与する」を参照してください。

IAM ロールと IAM ユーザーのどちらを使用するかについては、「IAM ユーザーガイド」の「IAM ロールの作成が適している場合 (ユーザーではなく)」を参照してください。

ポリシーを使用したアクセスの管理

でアクセスを制御する AWS には、ポリシーを作成し、 AWS ID またはリソースにアタッチします。ポリシーは のオブジェクト AWS であり、アイデンティティまたはリソースに関連付けられると、これらのアクセス許可を定義します。 は、プリンシパル (ユーザー、ルートユーザー、またはロールセッション) がリクエストを行うと、これらのポリシー AWS を評価します。ポリシーでの権限により、リクエストが許可されるか拒否されるかが決まります。ほとんどのポリシーは JSON ドキュメント AWS として に保存されます。JSON ポリシードキュメントの構造と内容の詳細については、「IAM ユーザーガイド」の「JSON ポリシー概要」を参照してください。

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

デフォルトでは、ユーザーやロールに権限はありません。IAM 管理者は、リソースで必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。その後、管理者はロールに IAM ポリシーを追加し、ユーザーはロールを引き継ぐことができます。

IAM ポリシーは、オペレーションの実行方法を問わず、アクションの権限を定義します。例えば、`iam:GetRole` アクションを許可するポリシーがあるとします。このポリシーを持つユーザーは、 AWS Management Console、、 AWS CLIまたは AWS API からロール情報を取得できます。

アイデンティティベースのポリシー

アイデンティティベースポリシーは、IAM ユーザー、ユーザーのグループ、ロールなど、アイデンティティにアタッチできる JSON 権限ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件を制御します。アイデンティティベースのポリシーを作成する方法については、「IAM ユーザーガイド」の「IAM ポリシーの作成」を参照してください。

アイデンティティベースのポリシーは、さらに インラインポリシー または マネージドポリシー に分類できます。インラインポリシーは、単一のユーザー、グループ、またはロールに直接埋め込まれています。管理ポリシーは、 の複数のユーザー、グループ、ロールにアタッチできるスタンドアロンポリシーです AWS アカウント。管理ポリシーには、 AWS 管理ポリシーとカスタマー管理ポリシーが含まれます。管理ポリシーまたはインラインポリシーのいずれかを選択する方法については、「IAM ユーザーガイド」の「管理ポリシーとインラインポリシーの比較」を参照してください。

リソースベースのポリシー

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。リソースベースのポリシーには例として、IAM ロールの信頼ポリシーや Amazon S3 バケットポリシーがあげられます。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスを制御できます。ポリシーが添付されているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。プリンシパルには、アカウント、ユーザー、ロール、フェデレーティッドユーザー、または を含めることができます AWS のサービス。

リソースベースのポリシーは、そのサービス内にあるインラインポリシーです。リソースベースのポリシーで IAM の AWS マネージドポリシーを使用することはできません。

アクセスコントロールリスト (ACL)

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするための権限を持つかをコントロールします。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

Amazon S3、、および Amazon VPC は AWS WAF、ACLs。ACL の詳細については、「Amazon Simple Storage Service デベロッパーガイド」の「アクセスコントロールリスト (ACL) の概要」を参照してください。

その他のポリシータイプ

AWS は、追加の一般的でないポリシータイプをサポートします。これらのポリシータイプでは、より一般的なポリシータイプで付与された最大の権限を設定できます。

- 権限の境界 - 権限の境界は、アイデンティティベースのポリシーによって IAM エンティティ (IAM ユーザーまたはロール) に付与できる権限の上限を設定する高度な機能です。エンティティに権限の境界を設定できます。結果として得られる権限は、エンティティのアイデンティティベースポリシーとその権限の境界の共通部分になります。`Principal` フィールドでユーザーまたはロールを指定するリソースベースのポリシーでは、権限の境界は制限されません。これらのポリシーのいずれかを明示的に拒否した場合、許可は無効になります。権限の境界の詳細については、「IAM ユーザーガイド」の「IAM エンティティの権限の境界」を参照してください。
- サービスコントロールポリシー (SCPs) – SCPs は、 の組織または組織単位 (OU) に最大アクセス許可を指定する JSON ポリシーです AWS Organizations。 AWS Organizations は、 AWS アカウント ビジネスが所有する複数の をグループ化して一元管理するためのサービスです。組織内のすべての機能を有効にすると、サービスコントロールポリシー (SCP) を一部またはすべてのアカウントに適用できます。SCP は、各 を含むメンバーアカウントのエンティティのアクセス許可を制限します AWS アカウントのルートユーザー。組織と SCP の詳細については、「AWS Organizations ユーザーガイド」の「SCP の仕組み」を参照してください。
- セッションポリシー - セッションポリシーは、ロールまたはフェデレーションユーザーの一時的なセッションをプログラムで作成する際にパラメータとして渡す高度なポリシーです。結果としてセッションの権限される範囲は、ユーザーまたはロールのアイデンティティベースポリシーとセッションポリシーの共通部分になります。また、リソースベースのポリシーから権限が派生する場合もあります。これらのポリシーのいずれかを明示的に拒否した場合、許可は無効になります。詳細については、「IAM ユーザーガイド」の「セッションポリシー」を参照してください。

複数のポリシータイプ

1 つのリクエストに複数のタイプのポリシーが適用されると、結果として作成される権限を理解するのがさらに難しくなります。複数のポリシータイプが関連する場合に がリクエストを許可するかどうか AWS を決定する方法については、IAM ユーザーガイドの「ポリシーの評価ロジック」を参照してください。

が IAM と AWS X-Ray 連携する方法

X-Ray へのアクセスを管理するために IAM を使用する前に、X-Ray でどの IAM 機能が使用できるかを理解しておく必要があります。X-Ray およびその他の が IAM と AWS のサービス 連携する方法の概要を把握するには、IAM AWS のサービス ユーザーガイドの「IAM と連携する」を参照してください。

AWS Identity and Access Management (IAM) を使用して、 アカウントのユーザーとコンピューティングリソースに X-Ray アクセス許可を付与できます。IAM は、ユーザーが採用するクライアント (コンソール、 AWS SDK AWS CLI) に関係なく、アクセス許可を統一的に適用するために、API レベルで X-Ray サービスへのアクセスを制御します。

X-Ray コンソールを使用してトレースマップとセグメントを表示するには、読み取りアクセス許可のみが必要です。コンソールアクセスを有効にするには、`AWSXrayReadOnlyAccess` 管理ポリシーを IAM ユーザーに追加します。

ローカルの開発とテストには、読み書きのアクセス許可を持つ IAM ロールを作成します。ロールを引き受け、そのロールの一時的な認証情報を保存します。これらの認証情報は、X-Ray デーモン、AWS CLI、および AWS SDK で使用できます。詳細については、「AWS CLIでの一時的なセキュリティ認証情報の使用」を参照してください。

計測済みアプリケーションを にデプロイするには AWS、書き込みアクセス許可を持つ IAM ロールを作成し、アプリケーションを実行しているリソースに割り当てます。`AWSXRayDaemonWriteAccess`には、トレースをアップロードするアクセス許可と、サンプリングルールの使用をサポートするいくつかの読み取りアクセス許可が含まれています。詳細については、「サンプリングルールを設定する」を参照してください。

読み書きポリシーには、暗号化キー設定とサンプリングルールを指定するためのアクセス許可は含まれていません。`AWSXrayFullAccess` を使用して、これらの設定にアクセスするか、カスタムポリシーに設定 API を追加します。作成したカスタマー管理キーで暗号化と複合を行うには、キーを使用するためのアクセス許可も必要です。

トピック

- X-Ray アイデンティティベースのポリシー
- X-Ray リソースベースのポリシー
- X-Ray タグに基づいた承認
- アプリケーションをローカルで実行する
- でのアプリケーションの実行 AWS
- 暗号化のユーザーアクセス許可

X-Ray アイデンティティベースのポリシー

IAM アイデンティティベースのポリシーでは、許可または拒否するアクションとリソース、アクションを許可または拒否する条件を指定できます。X-Ray は、特定のアクション、リソース、および条件キーをサポートしています。JSON ポリシーで使用するすべての要素については、「IAM ユーザーガイド」の「IAM JSON ポリシー要素のリファレンス」を参照してください。

アクション

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどんなリソースにどんな条件でアクションを実行できるかということです。

JSON ポリシーの`Action`要素には、ポリシー内のアクセスを許可または拒否するために使用できるアクションが記述されます。ポリシーアクションの名前は通常、関連付けられた AWS API オペレーションと同じです。一致する API オペレーションのない権限のみのアクションなど、いくつかの例外があります。また、ポリシーに複数アクションが必要なオペレーションもあります。これらの追加アクションは、 依存アクション と呼ばれます。

このアクションは、関連付けられたオペレーションを実行するための権限を付与するポリシーで使用されます。

X-Ray のポリシーアクションは、アクションの前に以下のプレフィックス を使用します: `xray:`。たとえば、X-Ray `GetGroup` API オペレーションを使用してグループリソースの詳細を取得するためのアクセス許可をユーザーに付与するには、ポリシーに `xray:GetGroup` アクションを含めます。ポリシーステートメントには、`Action` または `NotAction` 要素を含める必要があります。X-Ray は、このサービスで実行できるタスクを記述する独自のアクションのセットを定義します。

単一ステートメントに複数アクションを指定するには、次のようにカンマで区切ります:

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

```
      "xray:action2"
```

ワイルドカード (*) を使用して複数アクションを指定できます。たとえば、Get という単語で始まるすべてのアクションを指定するには、次のアクションを含めます。

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

X-Ray アクションのリストを表示するには、IAM ユーザーガイドの「AWS X-Rayによって定義されたアクション」を参照してください。

リソース

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースにどのような条件でアクションを実行できるかということです。

Resource JSON ポリシー要素は、アクションが適用されるオブジェクトを指定します。ステートメントには、Resource または NotResource要素を含める必要があります。ベストプラクティスとして、Amazon リソースネーム (ARN) を使用してリソースを指定します。これは、リソースレベルの権限と呼ばれる特定のリソースタイプをサポートするアクションに対して実行できます。

オペレーションのリスト化など、リソースレベルの権限をサポートしないアクションの場合は、ステートメントがすべてのリソースに適用されることを示すために、ワイルドカード (*) を使用します。

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

IAM ポリシーを使用してリソースへのアクセスを制御できます。リソースレベルのアクセス許可をサポートするアクションの場合は、Amazon リソースネーム (ARN) を使用して、ポリシーが適用されるリソースを識別します。

X-Ray ポリシーではすべての IAM アクションを使用して、そのアクションを使用するアクセス許可をユーザーに付与または拒否できます。ただし、すべての X-Ray アクションが、アクションを実行することができるリソースを指定できる、リソースレベルのアクセス許可をサポートしているわけではありません。

リソースレベルの権限をサポートしていないアクションの場合、「*」をリソースとして使用する必要があります。

次の X-Ray アクションは、リソースレベルのアクセス許可をサポートします。

- CreateGroup

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

以下は、CreateGroup アクションのアイデンティティベースのアクセス許可ポリシーの例です。この例では、グループ名 local-users に関連する ARN を使用する一意の ID をワイルドカードとして使用します。グループが作成されたときに一意の ID が生成されるため、事前にポリシーで予測することはできません。GetGroup、UpdateGroup、または DeleteGroup を使用する場合、ワイルドカードとして、または ID を含む正確な ARN として定義できます。

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

以下は、CreateSamplingRule アクションのアイデンティティベースのアクセス許可ポリシーの例です。

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

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

> **Note**
>
> サンプリングルールの ARN は、名前によって定義されます。グループ ARN とは異なり、サンプリングルールには一意に生成された ID がありません。

X-Ray リソースタイプとその ARN のリストを表示するには、IAM ユーザーガイドの「AWS X-Ray で定義されるリソース」を参照してください。どのアクションで各リソースの ARN を指定できるかについては、AWS X-Rayで定義されるアクションを参照してください。

条件キー

X-Ray にはサービス固有条件キーがありませんが、いくつかのグローバル条件キーの使用がサポートされています。すべての AWS グローバル条件キーを確認するには、「IAM ユーザーガイドAWS 」の「グローバル条件コンテキストキー」を参照してください。

例

X-Ray アイデンティティベースのポリシーの例を表示するには、「AWS X-Ray アイデンティティベースのポリシーの例」を参照してください。

X-Ray リソースベースのポリシー

X-Ray は、Amazon SNS アクティブトレースなど、現在および将来の AWS のサービス 統合のためのリソースベースのポリシーをサポートします。X-Ray のリソースベースのポリシーは、他の によって AWS Management Console、または AWS SDK または CLI を使用して更新できます。例えば、Amazon SNS コンソールは、X-Ray にトレースを送信するためのリソースベースのポリシーを自動的に設定しようとします。次のポリシードキュメントは、X-Ray リソースベースのポリシーを手動で設定する例を示しています。

Example Amazon SNS アクティブトレース用の X-Ray リソースベースのポリシーの例

このポリシードキュメントの例では、Amazon SNS がトレースデータを X-Ray に送信するために必要なアクセス許可を指定します。

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

CLI を使用して、Amazon SNS アクセス許可を付与するリソースベースのポリシーを作成して、トレースデータを X-Ray に送信します。

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

これらの例を使用するには、、*partition*、、 *topic-name*を特定の AWS パーティション*account-id*、リージョン、アカウント ID、および Amazon *region*SNS トピック名に置き換えます。 Amazon SNS すべての Amazon SNS トピックに、トレースデータを X-Ray に送信するアクセス許可を付与するには、トピック名を * に置き換えます。

X-Ray タグに基づいた承認

X-Ray グループまたはサンプリングルールにタグをアタッチしたり、リクエストでタグを X-Ray に渡すことができます。タグに基づいてアクセスを管理するには、`xray:ResourceTag/key-name`、`aws:RequestTag/key-name`、または `aws:TagKeys` の条件キーを使用して、ポリシーの条件要素でタグ情報を提供します。X-Ray リソースのタグ付けの詳細については、「X-Ray のサンプリングルールとグループのタグ付け」を参照してください。

リソースのタグに基づいてリソースへのアクセスを制限するためのアイデンティティベースポリシーの例を表示するには、「タグに基づいて X-Ray グループおよびサンプリングルールへのアクセスを管理する」を参照してください。

アプリケーションをローカルで実行する

実装したアプリケーションから X-Ray デーモンがトレースデータに送信されます。デーモンは、セグメントドキュメントをバッファし、バッチ処理で X-Ray サービスにアップロードします。デーモンは、X-Ray サービスにトレースデータおよびテレメトリをアップロードする書き込みのアクセス許可が必要です。

デーモンをローカルで実行する場合は、IAM ロールを作成し、そのロールを引き受けて、一時的な認証情報を環境変数に保存するか、ユーザーフォルダー内の `.aws` フォルダーの `credentials` ファイルに保存します。詳細については、「AWS CLIでの一時的なセキュリティ認証情報の使用」を参照してください。

Example ~/.aws/credentials

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

AWS SDK または で使用する認証情報をすでに設定している場合 AWS CLI、デーモンはそれらを使用できます。複数のプロファイルが利用可能な場合はデフォルトのプロファイルが使用されます。

でのアプリケーションの実行 AWS

でアプリケーションを実行するときは AWS、ロールを使用して、デーモンを実行する Amazon EC2 インスタンスまたは Lambda 関数にアクセス許可を付与します。

- Amazon Elastic Compute Cloud (Amazon EC2) – IAM ロールを作成し、「インスタンスプロファイル」として EC2 インスタンスにアタッチします。

- Amazon Elastic Container Service (Amazon ECS) – IAM ロールを作成し、コンテナインスタンスの IAM ロールとしてコンテナインスタンスにアタッチします。
- AWS Elastic Beanstalk (Elastic Beanstalk) – Elastic Beanstalk には、デフォルトのインスタンスプロファイル に X-Ray アクセス許可が含まれます。デフォルトのインスタンスプロファイルを使用するか、カスタムのインスタンスプロファイルに書き込みのアクセス許可を追加できます。
- AWS Lambda (Lambda) – 関数の実行ロールに書き込みアクセス許可を追加します。

X-Ray で使用するためのロールを作成するには

1. [IAM コンソール] を開きます。
2. [ロール] を選択します。
3. [Create New Role (新しいロールを作成)] を選択します。
4. [Role Name (ロール名)] に **`xray-application`** を入力します。[Next Step (次のステップ)] をクリックします。
5. [Role Type (ロールタイプ)] で、[Amazon EC2] を選択します。
6. 次の管理ポリシーをアタッチして AWS のサービスへのアクセス権限をアプリケーションに付与します。

   - AWSXRayDaemonWriteAccess – トレースデータをアップロードするアクセス許可を X-Ray デーモンに付与します。

   アプリケーションが AWS SDK を使用して他の サービスにアクセスする場合は、それらのサービスへのアクセスを許可するポリシーを追加します。
7. [Next Step (次のステップ)] をクリックします。
8. [Create Role (ロールを作成)] を選択します。

暗号化のユーザーアクセス許可

X-Ray は、すべてのトレースデータを暗号化します。デフォルトでは、管理するキーを使用するように設定できます。 AWS Key Management Service カスタマーマネージドキーを選択した場合は、キーのアクセスポリシーで、暗号化に使用するアクセス許可を X-Ray に付与できることを確認する必要があります。また、アカウントの他のユーザーがキーにアクセスし、X-Ray コンソールで暗号化されたトレースデータを確認できるようにする必要があります。

カスタマー管理キーに関しては、以下のアクションが可能なアクセスポリシーを使用してキーを設定します。

- X-Ray でキーを設定するユーザーには、`kms:CreateGrant` と `kms:DescribeKey` を呼び出すためのアクセス許可があります。
- 暗号化されたトレースデータにアクセスできるユーザーには、`kms:Decrypt` を呼び出すためのアクセス許可があります。

IAM コンソールのキー設定セクション内の主要ユーザーグループにユーザーを追加する場合、彼らにはこれらの両方のオペレーションに対するアクセス許可があります。アクセス許可はキーポリシーでのみ設定する必要があるため、ユーザー、グループ、またはロールに対する AWS KMS アクセス許可は必要ありません。詳細については、「AWS KMS デベロッパーガイド」の「キーポリシーの使用」を参照してください。

デフォルトの暗号化の場合、または AWS マネージド CMK (`aws/xray`) を選択した場合、アクセス許可は X-Ray APIs。`AWSXrayFullAccess` に含まれる `PutEncryptionConfig`にアクセスできるユーザーはすべて、暗号化の設定を変更することが可能です。ユーザーが暗号化キーを変更できないようにする場合は、`PutEncryptionConfig` を使用するためのアクセス許可を付与しないようにしてください。

AWS X-Ray アイデンティティベースのポリシーの例

デフォルトでは、ユーザーおよびロールには、X-Ray リソースを作成または変更するアクセス許可はありません。また、、 AWS Management Console AWS CLI、または AWS API を使用してタスクを実行することはできません。IAM 管理者は、指定されたリソースで特定の API 操作を実行するための許可をユーザーとロールに付与する IAM ポリシーを作成する必要があります。続いて、管理者はそれらのアクセス許可が必要なユーザーまたはグループにそのポリシーをアタッチします。

JSON ポリシードキュメントのこれらの例を使用して、IAM アイデンティティベースのポリシーを作成する方法については、「IAM ユーザーガイド」の「JSON タブでのポリシーの作成」を参照してください。

トピック

- ポリシーのベストプラクティス
- X-Ray コンソールの使用
- ユーザーが自分の許可を表示できるようにする
- タグに基づいて X-Ray グループおよびサンプリングルールへのアクセスを管理する

- X-Ray の IAM マネージドポリシー
- X-Ray での AWS マネージドポリシーの更新
- IAM ポリシーでリソースを指定する

ポリシーのベストプラクティス

ID ベースのポリシーは、ユーザーのアカウントで誰が X-Ray リソースを作成、アクセス、削除できるかを決定します。これらのアクションを実行すると、 AWS アカウントに料金が発生する可能性があります。アイデンティティベースポリシーを作成したり編集したりする際には、以下のガイドラインと推奨事項に従ってください:

- AWS 管理ポリシーを開始し、最小特権のアクセス許可に移行する – ユーザーとワークロードにアクセス許可を付与するには、多くの一般的なユースケースにアクセス許可を付与する AWS 管理ポリシーを使用します。これらは で使用できます AWS アカウント。ユースケースに固有の AWS カスタマー管理ポリシーを定義して、アクセス許可をさらに減らすことをお勧めします。詳細については、IAM ユーザーガイドの「AWS マネージドポリシー」または「AWS ジョブ機能の管理ポリシー」を参照してください。
- 最小特権を適用する – IAM ポリシーで権限を設定するときは、タスクの実行に必要な権限のみを付与します。これを行うには、特定の条件下で特定のリソースに対して実行できるアクションを定義します。これは、最小特権権限とも呼ばれています。IAM を使用して権限を適用する方法の詳細については、『IAM ユーザーガイド』の「IAM でのポリシーと権限」を参照してください。
- IAM ポリシーで条件を使用してアクセスをさらに制限する - ポリシーに条件を追加して、アクションやリソースへのアクセスを制限できます。例えば、ポリシー条件を記述して、すべてのリクエストを SSL を使用して送信するように指定できます。条件を使用して、 などの特定の を介してサービスアクションが使用される場合に AWS のサービス、サービスアクションへのアクセスを許可することもできます AWS CloudFormation。詳細については、IAM ユーザーガイドの [IAM JSON policy elements: Condition] (IAM JSON ポリシー要素：条件) を参照してください。
- IAM Access Analyzer を使用して IAM ポリシーを検証し、安全で機能的な権限を確保する - IAM Access Analyzer は、新規および既存のポリシーを検証して、ポリシーが IAM ポリシー言語 (JSON) および IAM のベストプラクティスに準拠するようにします。IAM アクセスアナライザーは 100 を超えるポリシーチェックと実用的な推奨事項を提供し、安全で機能的なポリシーの作成をサポートします。詳細については、「IAM ユーザーガイド」の「IAM Access Analyzer ポリシーの検証」を参照してください。
- 多要素認証 (MFA) を要求する – で IAM ユーザーまたはルートユーザーを必要とするシナリオがある場合は AWS アカウント、セキュリティを強化するために MFA を有効にします。API オペレー

ションが呼び出されるときに MFA を必須にするには、ポリシーに MFA 条件を追加します。詳細については、「IAM ユーザーガイド」の「MFA 保護 API アクセスの設定」を参照してください。

IAM でのベストプラクティスの詳細については、『IAM ユーザーガイド』の「IAM でのセキュリティのベストプラクティス」を参照してください。

X-Ray コンソールの使用

AWS X-Ray コンソールにアクセスするには、最小限のアクセス許可のセットが必要です。これらのアクセス許可により、 の X-Ray リソースの詳細を一覧表示および表示できます AWS アカウント。最小限必要な許可よりも制限が厳しいアイデンティティベースのポリシーを作成すると、そのポリシーを持つエンティティ (ユーザーまたはロール) に対してコンソールが意図したとおりに機能しません。

これらのエンティティが X-Ray コンソールを引き続き使用できるようにするには、エンティティに AWSXRayReadOnlyAccess AWS 管理ポリシーをアタッチします。このポリシーについては、X-Ray の IAM マネージドポリシーで詳しく説明されています。詳細については、「IAM ユーザーガイド」の「ユーザーへのアクセス許可の追加」を参照してください。

AWS CLI または AWS API のみを呼び出すユーザーには、最小限のコンソールアクセス許可を付与する必要はありません。代わりに、実行しようとしている API オペレーションに一致するアクションのみへのアクセスが許可されます。

ユーザーが自分の許可を表示できるようにする

この例では、ユーザーアイデンティティにアタッチされたインラインおよびマネージドポリシーの表示を IAM ユーザーに許可するポリシーの作成方法を示します。このポリシーには、コンソールで、または AWS CLI または AWS API を使用してプログラムでこのアクションを実行するアクセス許可が含まれています。

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

タグに基づいて X-Ray グループおよびサンプリングルールへのアクセスを管理する

アイデンティティベースのポリシーの条件を使用して、タグに基づいて X-Ray グループやタグに基づいたサンプリングルールへのアクセスを制御できます。次の例のポリシーはタグ `stage:prod` または `stage:preprod` 付きのグループを作成、削除、または更新するアクセス権限をユーザーロールで拒否するために使用できます。X-Ray サンプリングルールとグループのタグ付けの詳細については、「X-Ray のサンプリングルールとグループのタグ付け」を参照してください。

ユーザーがタグ `stage:prod` または `stage:preprod` 付きのグループの作成、更新、または削除にアクセスするのを拒否するには、次のようなポリシーを持つロールをユーザーに割り当てます。

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

サンプリングルールの作成を拒否するには、aws:RequestTag を使って作成リクエストの一部として渡すことができないタグを示します。サンプリングルールの更新または削除を拒否するには、aws:ResourceTag を使って、それらのリソースのタグに基づくアクションを拒否します。

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

これらのポリシーをアカウントのユーザーにアタッチする (または、単一のポリシーに結合してからポリシーをアタッチする) ことができます。ユーザーがグループまたはサンプリングルールに変更を加えるには、グループまたはサンプリングルールにタグ `stage=prepod` または `stage=prod`

を付けないでください。条件キー名では大文字と小文字が区別されないため、条件タグキー Stage は Stage と stage の両方に一致します。条件ブロックの詳細については、IAM ユーザーガイドの「IAM JSON ポリシーの要素: 条件」を参照してください。

次のポリシーがアタッチされているロールを持つユーザーは、タグ role:admin を追加してリソースにアクセスしたり、role:admin に関連付けられたリソースからタグを削除したりすることはできません。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAllXRay",
            "Effect": "Allow",
            "Action": "xray:*",
            "Resource": "*"
        },
        {
            "Sid": "DenyRequestTagAdmin",
            "Effect": "Deny",
            "Action": "xray:TagResource",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/role": "admin"
                }
            }
        },
        {
            "Sid": "DenyResourceTagAdmin",
            "Effect": "Deny",
            "Action": "xray:UntagResource",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/role": "admin"
                }
            }
        }
    ]
}
```

X-Ray の IAM マネージドポリシー

アクセス許可を簡単に付与するために、IAM は各サービスのマネージドポリシーに対応しています。サービスは、新しい APIs。 AWS X-Ray は、読み取り専用、書き込み専用、管理者のユースケース用の管理ポリシーを提供します。

- `AWSXrayReadOnlyAccess` – X-Ray コンソール、または AWS SDK を使用して AWS CLI、X-Ray API からトレースデータ、トレースマップ、インサイト、X-Ray 設定を取得するための読み取りアクセス許可。オブザーバビリティアクセスマネージャー (OAM) `oam:ListSinks`と、CloudWatchクロスアカウントオブザーバビリティ の一部としてソースアカウントから共有されたトレースをコンソールが表示できるようにする`oam:ListAttachedSinks`アクセス許可が含まれています。`BatchGetTraceSummaryById` および `GetDistinctTraceGraphs` API アクションは、コードによって呼び出されることを意図したものではなく、 AWS CLI および AWS SDKsには含まれていません。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "xray:GetSamplingRules",
                "xray:GetSamplingTargets",
                "xray:GetSamplingStatisticSummaries",
                "xray:BatchGetTraces",
                "xray:BatchGetTraceSummaryById",
                "xray:GetDistinctTraceGraphs",
                "xray:GetServiceGraph",
                "xray:GetTraceGraph",
                "xray:GetTraceSummaries",
                "xray:GetGroups",
                "xray:GetGroup",
                "xray:ListTagsForResource",
                "xray:ListResourcePolicies",
                "xray:GetTimeSeriesServiceStatistics",
                "xray:GetInsightSummaries",
                "xray:GetInsight",
                "xray:GetInsightEvents",
                "xray:GetInsightImpactGraph",
                "oam:ListSinks"
            ],
            "Resource": [
```

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

}
```

- `AWSXRayDaemonWriteAccess` – X-Ray デーモン AWS CLI、または AWS SDK を使用してセグメントドキュメントとテレメトリを X-Ray API にアップロードするための書き込みアクセス許可。サンプリングルールを取得してサンプリング結果を報告するための読み取りアクセス許可が含まれています。詳細については、「サンプリングルールを設定する」を参照してください。

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

- `AWSXrayCrossAccountSharingConfiguration` – アカウント間で X-Ray リソースを共有するための Observability Access Manager のリンクを作成、管理、表示するためのアクセス許可を付与します。ソースCloudWatch アカウントとモニタリングアカウント間のクロスアカウントオブザーバビリティを有効にするために使用されます。

```
{
    "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` – 読み取りアクセス許可、書き込みアクセス許可、および暗号化キー設定とサンプリングルールを指定するためのアクセス許可を含む、すべての X-Ray API を使用するためのアクセス許可。オブザーバビリティアクセスマネージャー (OAM) `oam:ListSinks`と、CloudWatchクロスアカウントオブザーバビリティ の一部としてソースアカウントから共有されたトレースをコンソールが表示できるようにする`oam:ListAttachedSinks`アクセス許可が含まれています。

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

マネージドポリシーを IAM ユーザー、グループ、ロールに追加するには

1. [IAM コンソール] を開きます。
2. インスタンスプロファイル、IAM ユーザー、または IAM グループに関連付けられたロールを開きます。
3. [アクセス許可] で、管理ポリシーをアタッチします。

X-Ray での AWS マネージドポリシーの更新

X-Ray の AWS マネージドポリシーの更新に関する詳細を、このサービスがこれらの変更の追跡を開始した以降の分について表示します。このページの変更に関する自動通知については、X-Ray ドキュメントの履歴ページの RSS フィードをサブスクライブしてください。

変更	説明	日付
X-Ray 用の IAM マネージドポリシー – 新しいポリシー AWSXrayCrossAccountSharingConfiguration を追加、AWSXrayReadOnlyAccess と AWSXrayFullAccess を更新	X-Ray は、オブザーバビリティアクセスマネージャー (OAM) のアクセス許可oam:ListSinks と oam:ListAttachedSinks をこれらのポリシーに追加し、コンソールがCloudWatch クロスアカウントオブザーバビリティ の一部としてソースアカウントから共有されたトレースを表示できるようにしました。	2022 年 11 月 27 日
X-Ray の IAM マネージドポリシー – AWSXrayReadOnlyAccess ポリシーの更新	X-Ray に API アクション ListResourcePolicies が追加されました。	2022 年 11 月 15 日
X-Ray コンソールの使用 – AWSXrayReadOnlyAccess ポリシーの更新	X-Ray に BatchGetTraceSummaryById と GetDistinctTraceGraphs の 2 つの新しい API アクションが追加されました。 これらのアクションは、コードで呼び出すものではありません。したがって、これらの API アクションは AWS CLI および AWS SDKsには含まれていません。	2022 年 11 月 11 日

IAM ポリシーでリソースを指定する

IAM ポリシーを使用してリソースへのアクセスを制御できます。リソースレベルのアクセス許可をサポートするアクションの場合は、Amazon リソースネーム (ARN) を使用して、ポリシーが適用されるリソースを識別します。

X-Ray ポリシーではすべての IAM アクションを使用して、そのアクションを使用するアクセス許可をユーザーに付与または拒否できます。ただし、すべての X-Ray アクションが、アクションを実行することができるリソースを指定できる、リソースレベルのアクセス許可をサポートしているわけではありません。

リソースレベルの権限をサポートしていないアクションの場合、「*」をリソースとして使用する必要があります。

次の X-Ray アクションは、リソースレベルのアクセス許可をサポートします。

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

以下は、`CreateGroup` アクションのアイデンティティベースのアクセス許可ポリシーの例です。この例では、グループ名 `local-users` に関連する ARN を使用する一意の ID をワイルドカードとして使用します。グループが作成されたときに一意の ID が生成されるため、事前にポリシーで予測することはできません。`GetGroup`、`UpdateGroup`、または `DeleteGroup` を使用する場合、ワイルドカードとして、または ID を含む正確な ARN として定義できます。

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

```
            "Resource": [
                "arn:aws:xray:eu-west-1:123456789012:group/local-users/*"
            ]
        }
    ]
}
```

以下は、CreateSamplingRule アクションのアイデンティティベースのアクセス許可ポリシーの例です。

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

> **Note**
>
> サンプリングルールの ARN は、名前によって定義されます。グループ ARN とは異なり、サンプリングルールには一意に生成された ID がありません。

AWS X-Ray アイデンティティとアクセスに関するトラブルシューティング

次の情報は、X-Ray と IAM の使用に伴って発生する可能性がある一般的な問題の診断や修復に役立ちます。

トピック

- X-Ray でアクションを実行する権限がない
- iam:PassRole を実行する認可がない
- 管理者として X-Ray へのアクセスを他のユーザーに許可したい

- 自分の AWS アカウント 以外のユーザーに X-Ray リソースへのアクセスを許可したい

X-Ray でアクションを実行する権限がない

AWS Management Console から、アクションを実行することが認可されていないと通知された場合、管理者に問い合わせ、サポートを依頼する必要があります。管理者は、サインイン認証情報を提供した担当者です。

次の例のエラーは、mateojackson ユーザーがコンソールを使用してサンプリングルールの詳細を表示する際に、xray:GetSamplingRules アクセス許可を持っていない場合に発生します。

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

この場合、Mateo は管理者に依頼し、xray:GetSamplingRules アクションを使用してサンプリングルールリソースにアクセスできるようにポリシーを更新してもらいます。

iam:PassRole を実行する認可がない

iam:PassRole アクションを実行する権限がないというエラーが表示された場合は、ポリシーを更新して X-Ray にロールを渡すことができるようにする必要があります。

一部の AWS のサービス では、新しいサービスロールまたはサービスにリンクされたロールを作成せずに、既存のロールをサービスに渡すことが許可されています。そのためには、サービスにロールを渡す許可が必要です。

以下の例のエラーは、marymajor という IAM ユーザーがコンソールを使用して X-Ray でアクションを実行しようする場合に発生します。ただし、このアクションをサービスが実行するには、サービスロールから付与されたアクセス許可が必要です。Mary には、ロールをサービスに渡す許可がありません。

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

この場合、メアリーのポリシーを更新してメアリーに iam:PassRole アクションの実行を許可する必要があります。

サポートが必要な場合は、AWS 管理者に問い合わせてください。サインイン資格情報を提供した担当者が管理者です。

管理者として X-Ray へのアクセスを他のユーザーに許可したい

X-Ray へのアクセスを他のユーザーに許可するには、アクセスを必要とする人またはアプリケーションの IAM エンティティ (ユーザーまたはロール) を作成する必要があります。ユーザーまたはアプリケーションは、このエンティティの認証情報を使用して AWS にアクセスします。次に、X-Ray の適切なアクセス許可を付与するポリシーを、そのエンティティにアタッチする必要があります

すぐにスタートするには、「IAM ユーザーガイド」の「IAM が委任した初期のユーザーおよびグループの作成」を参照してください。

自分の AWS アカウント 以外のユーザーに X-Ray リソースへのアクセスを許可したい

他のアカウントのユーザーや組織外のユーザーが、リソースにアクセスするために使用できるロールを作成できます。ロールの引き受けを委託するユーザーを指定することができます。リソースベースのポリシーまたはアクセス制御リスト (ACL) をサポートするサービスの場合、それらのポリシーを使用して、リソースへのアクセスを付与できます。

詳細については、以下を参照してください。

- X-Ray がこれらの機能をサポートしているかどうかを確認するには、「が IAM と AWS X-Ray 連携する方法」を参照してください。
- 所有している AWS アカウント 全体のリソースへのアクセス権を提供する方法については、「IAM ユーザーガイド」の「所有している別の AWS アカウント アカウントへのアクセス権を IAM ユーザーに提供」を参照してください。
- サードパーティーの AWS アカウント にリソースへのアクセス権を提供する方法については、「IAM ユーザーガイド」の「第三者が所有する AWS アカウント へのアクセス権を付与する」を参照してください。
- ID フェデレーションを介してアクセスを提供する方法については、「IAM ユーザーガイド」の「外部で認証されたユーザー (ID フェデレーション) へのアクセスの許可」を参照してください。
- クロスアカウントアクセスでのロールとリソースベースのポリシーの使用の違いの詳細については、「IAM ユーザーガイド」の「IAM ロールとリソースベースのポリシーとの相違点」を参照してください。

AWS X-Rayでのログ記録とモニタリング

モニタリングは、AWSソリューションの信頼性、可用性、および性能を維持するうえで重要な部分です。マルチポイントの障害が発生した場合は、より簡単に欠陥を捜して直すことができる

ように、AWSソリューションの、すべての部分からモニタリングデータを収集する必要があります。AWSそれには、X-Rayリソースをモニタリングし、潜在的な事態に対応するため、複数のツールが用意されています：

AWS CloudTrailログ

AWS X-Rayは、AWS CloudTrailX-Rayのユーザーまたは、AWS役割、サービスによって行われた API操作を記録するために統合されています。CloudTrailを使用して、X-Ray API依頼を即時にモニタリングし、Amazon S3、Amazon CloudWatch Logs、Amazon CloudWatch Eventsにログを保存することができます。詳細については、「を使用した X-Ray API コールのログ記録 AWS CloudTrail」を参照してください。

AWS Config追跡

AWS X-Rayと統合することで、AWS ConfigX-Ray暗号化リソースに対して行われた設定変更を記録することができます。X-Rayの暗号化リソースの一覧表を使用できます、AWS ConfigX-Rayの設定履歴の監査、およびリソースの変更に基づいて通知を送信することができます。詳細については、「X-Rayの暗号化設定の変更を追跡AWS Config」を参照してください。

Amazon CloudWatchのモニタリング

X-Ray SDK for Java を使用して、収集した X-Rayセグメントからサンプリングされずに Amazon CloudWatch メトリクスを公開することができます。これらのメトリクスは、セグメントの開始時間と終了時間、さらにエラー、障害およびスロットル状態を示すフラグから取得されます。これらの追跡メトリクスを使用して、サブセグメント内の再試行と依存関係の問題を公開します。詳細については、「AWS X-Ray X-Ray SDK for Java の メトリクス」を参照してください。

のコンプライアンス検証 AWS X-Ray

AWS のサービス が特定のコンプライアンスプログラムの範囲内にあるかどうかを確認するには、コンプライアンスプログラムAWS のサービス による対象範囲内のコンプライアンスプログラムを参照し、関心のあるコンプライアンスプログラムを選択します。一般的な情報については、AWS 「コンプライアンスプログラム」を参照してください。

を使用して、サードパーティーの監査レポートをダウンロードできます AWS Artifact。詳細については、「 でのレポートのダウンロード AWS Artifact」の」を参照してください。

を使用する際のお客様のコンプライアンス責任 AWS のサービス は、お客様のデータの機密性、貴社のコンプライアンス目的、適用される法律および規制によって決まります。 では、コンプライアンスに役立つ以下のリソース AWS を提供しています。

- セキュリティとコンプライアンスのクイックスタートガイド – これらのデプロイガイドでは、アーキテクチャ上の考慮事項について説明し、セキュリティとコンプライアンスに重点を置いたベースライン環境 AWS を にデプロイする手順について説明します。
- アマゾン ウェブ サービスにおける HIPAA セキュリティとコンプライアンスのアーキテクチャ — このホワイトペーパーでは、企業が AWS を使用して HIPAA 対象アプリケーションを作成する方法について説明します。

> **Note**
>
> すべて AWS のサービス HIPAA の対象となるわけではありません。詳細については、「HIPAA 対応サービスのリファレンス」を参照してください。

- AWS コンプライアンスリソース – このワークブックとガイドのコレクションは、お客様の業界や地域に適用される場合があります。
- AWS カスタマーコンプライアンスガイド – コンプライアンスの観点から責任共有モデルを理解します。このガイドでは、ガイダンスを保護し AWS のサービス 、複数のフレームワーク (米国国立標準技術研究所 (NIST)、Payment Card Industry Security Standards Council (PCI)、国際標準化機構 (ISO) を含む) のセキュリティコントロールにマッピングするためのベストプラクティスをまとめています。
- 「 デベロッパーガイド」の「 ルールによるリソースの評価」 – この AWS Config サービスは、リソース設定が社内プラクティス、業界ガイドライン、および規制にどの程度準拠しているかを評価します。 AWS Config
- AWS Security Hub – これにより AWS のサービス 、 内のセキュリティ状態を包括的に把握できます AWS。Security Hub では、セキュリティコントロールを使用して AWS リソースを評価し、セキュリティ業界標準とベストプラクティスに対するコンプライアンスをチェックします。サポートされているサービスとコントロールのリストについては、「Security Hub のコントロールリファレンス」を参照してください。
- Amazon GuardDuty – これにより AWS アカウント、疑わしいアクティビティや悪意のあるアクティビティがないか環境を監視することで、、、ワークロード、コンテナ、データに対する潜在的な脅威 AWS のサービス を検出します。 GuardDuty は、特定のコンプライアンスフレームワークで義務付けられている侵入検知要件を満たすことで、PCI DSS などのさまざまなコンプライアンス要件への対応に役立ちます。
- AWS Audit Manager – これにより AWS のサービス 、 AWS 使用状況を継続的に監査し、リスクの管理方法と規制や業界標準への準拠を簡素化できます。

AWS X-Ray での耐障害性

AWS グローバルインフラストラクチャは AWS リージョン およびアベイラビリティーゾーンを中心に構築されています。AWS リージョン には、低レイテンシー、高いスループット、そして高度の冗長ネットワークで接続されている複数の物理的に独立· 隔離されたアベイラビリティーゾーンがあります。アベイラビリティーゾーンでは、アベイラビリティーゾーン間で中断せずに、自動的にフェイルオーバーするアプリケーションとデータベースを設計および運用することができます。アベイラビリティーゾーンは、従来の単一または複数のデータセンターインフラストラクチャに比べて、可用性、耐障害性、および拡張性に優れています。

AWS リージョン とアベイラビリティーゾーンの詳細については、「AWS グローバルインフラストラクチャ」を参照してください。

AWS X-Ray でのインフラストラクチャセキュリティ

マネージドサービスである AWS X-Ray は AWS グローバルネットワークセキュリティで保護されています。AWSセキュリティサービスと AWS がインフラストラクチャを保護する方法については、「AWS クラウドセキュリティ」を参照してください。インフラストラクチャセキュリティのベストプラクティスを使用して AWS 環境を設計するには、「セキュリティの柱 - AWS Well-Architected Framework」の「インフラストラクチャ保護」を参照してください。

AWSネットワーク経由でX-Rayにアクセスするには、公開されたAPIコールを使用します。クライアントは以下をサポートする必要があります。

- Transport Layer Security (TLS) TLS 1.2 および TLS 1.3 をお勧めします。
- DHE (Ephemeral Diffie-Hellman) や ECDHE (Elliptic Curve Ephemeral Diffie-Hellman) などの Perfect Forward Secrecy (PFS) を使用した暗号スイートです。これらのモードは、Java 7 以降など、最近のほとんどのシステムでサポートされています。

また、リクエストは、アクセスキー ID と、IAM プリンシパルに関連付けられているシークレットアクセスキーを使用して署名する必要があります。または、AWS Security Token Service (AWS STS) を使用して、一時的なセキュリティ認証情報を生成し、リクエストに署名することもできます。

VPC評価項目でAWS X-Rayを使用する

Amazon Virtual Private Cloud(Amazon VPC)AWSを使用してリソースを主催している場合、VPCおよびX-Rayの間にプライベート接続を確立することができます。これによりAmazon VPC内のリソースが公衆インターネットを経由せずに、X-Rayサービスと通信できるようになります。

Amazon VPC は、ユーザー定義の仮想ネットワークで AWS リソースを起動するために使用できる AWS のサービス です。VPC を使用すると、IP アドレス範囲、サブネット、ルートテーブル、ネットワークゲートウェイなどのネットワーク設定を制御できます。VPCをX-Rayに接続するには、インターフェイスのVPC評価項目を定義します。この評価項目は、インターネットゲートウェイ、ネットワークアドレス変換(NAT)の場合、またはVPN接続を必要とせず、X-Rayへの信頼性が高く拡張性のある接続性を提供します。詳細については、『Amazon VPC ユーザーガイド』の「Amazon VPC とは何か」を参照してください。

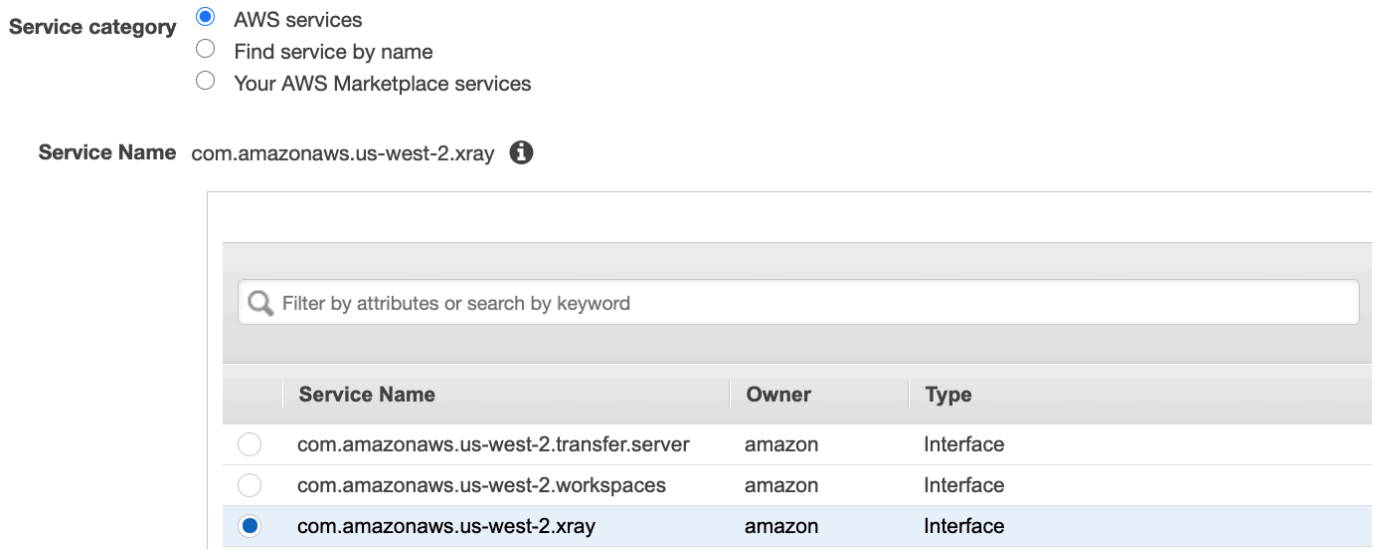
インターフェイス VPC エンドポイントは AWS PrivateLink を使用しています。これは、Elastic Network Interface とプライベート IP アドレスを使用して AWS のサービス の間のプライベート通信を可能にする AWS のテクノロジーです。詳細については、New – AWS PrivateLink for AWS のサービス ブログ投稿と「Amazon VPC ユーザーガイド」の「はじめに」を参照してください。

選択した AWS リージョン で X-Ray の VPC エンドポイントを確実に作成できるようにするためにはサポートされている地域を参照してください。

X-Ray用のVPC評価項目の作成

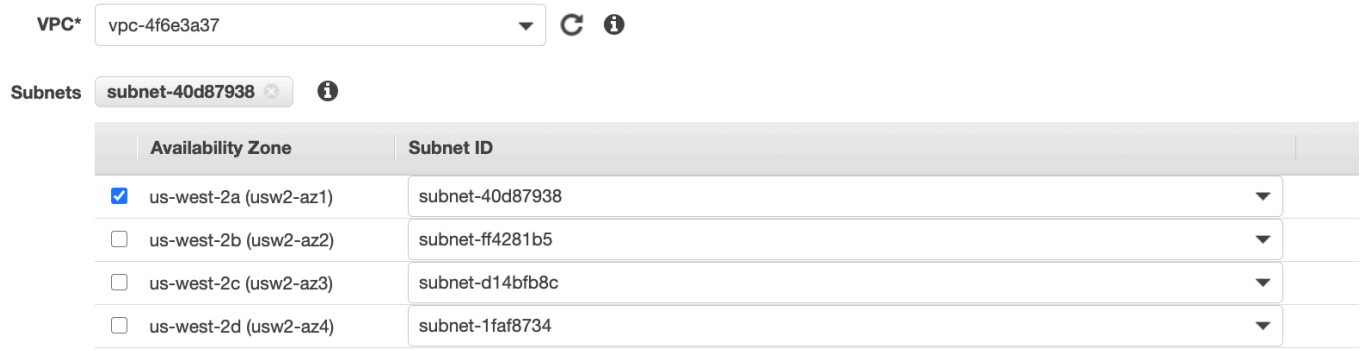
VPCでX-Rayの使用を開始するには、X-Rey用のインターフェイスVPC評価項目を作成します。

1. (https://console.aws.amazon.com/vpc/)Amazon VPC制御盤を開きます。
2. ナビゲーションペイン内の、「評価項目」に移動して評価項目の作成を選択します。
3. サービス名を検索して選択してください。AWS X-Raycom.amazonaws.*region*.xray

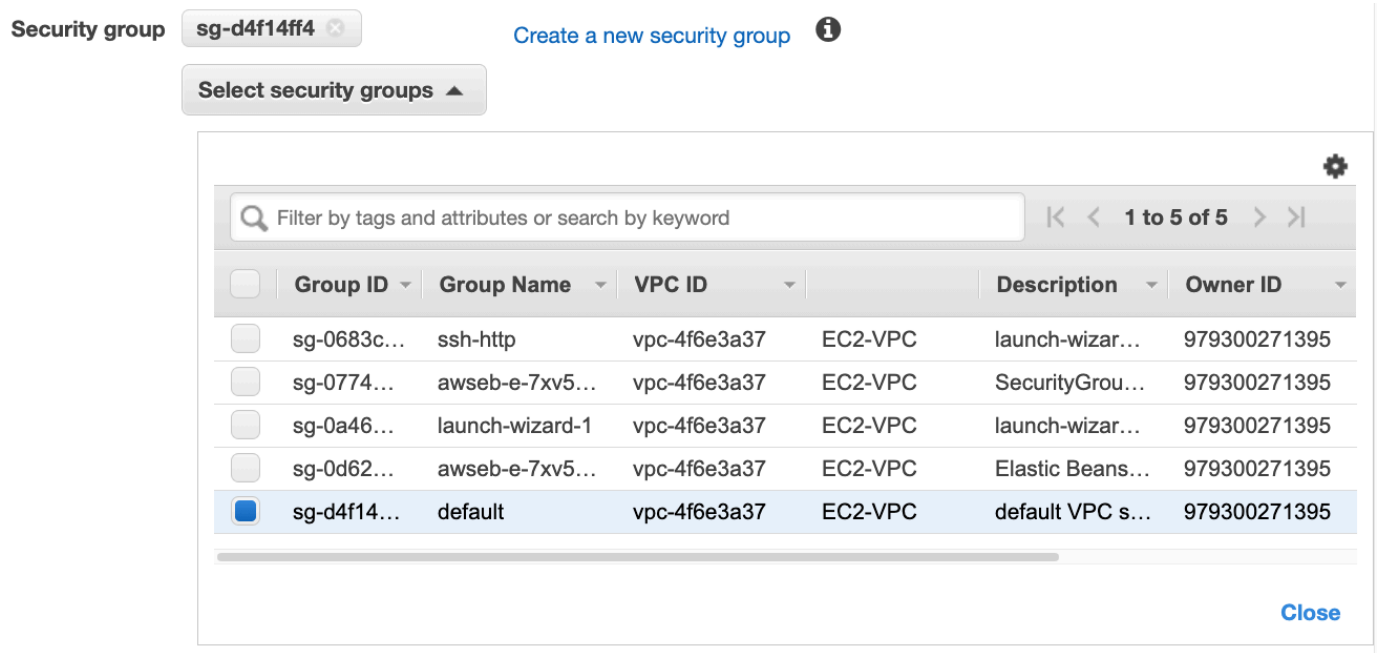


4. インターフェイス評価項目を使用して目的のVPCを選択し、VPC内のサブネットを選択します。選択したサブネットに評価項目のネットワークインターフェイスが作成されます。サービスでサポートされている異なる可用性の範囲に複数のサブネットを指定することで、可用性の範囲

の障害に対応したインターフェイス評価項目を、より確実に回復できます。この操作を行うと、指定した各サブネットでインターフェイスのネットワークインターフェイスが作成されます。



5. (オプション)初期設定の情報システムのDNSホスト名を使用してX-Rayのリクエストを行うには、評価項目のDNSプライベート情報システムが初期設定で有効になっています。無効にすることも可能です。
6. 評価項目ネットワークインターフェイスに関連付けるセキュリティグループを指定します。



7. （オプション）X-Rayサービスへのアクセス許可を制御するカスタムポリシーを指定します。初期設定では、数多くのアクセスが許可されています。

X-RayVPCの情報システム評価項目へのアクセスの制御

VPC評価項目ポリシーは、評価項目の作成時または変更時に評価項目に加える国際機械技術者協会（IAM）のリソースポリシーです。評価項目の作成時にポリシーを加えない場合、サービスへの数多

くのアクセスを許可する初期設定のポリシーがAmazon VPCによって自動的に接続されます。評価項目ポリシーは、IAMユーザーポリシーやサービス固有のポリシーを上書き、または置き換えたりするものではありません。これは、評価項目から指定されたサービスへのアクセスを制御するための別のポリシーです。評価項目のポリシーは、JSON形式で記載する必要があります。詳細については、「Amazon VPCユーザーガイド」の「VPC評価項目によるサービスのアクセス制御」を参照してください。

VPC評価項目ポリシーを使用すると、さまざまなX-Ray行為に対するアクセス許可を制御することができます。たとえば、putTraceSegmentのみを許可し、その他のすべての行為を拒否するポリシーを作成することができます。これにより、VPC内のワークロードおよびサービスは、X-Rayにトレースデータのみを送信し、データの取得、暗号化設定の変更、グループの作成/更新など、その他の行為を拒否するように制限されます。

X-Rayの評価項目ポリシーの例を以下に示します。このポリシーは、VPCを介してX-Rayに接続するユーザーに対して、セグメントデータをX-Reyに送信することを許可し、また他のX-Rey行為を実行することを禁止します。

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

X-RayのVPC評価項目ポリシーを編集するには

1. Amazon VPCコンソール(https://console.aws.amazon.com/vpc/)を開きます。
2. ナビゲーションペインで、[評価項目]を選択します。
3. X-Ray用の評価項目を、まだ作成していない場合は、X-Ray用のVPC評価項目の作成の手順に従います。
4. [com.amazonaws.##.xray]評価項目を選択し、[ポリシー]タブを選択します。
5. [ポリシーの編集]を選択し、変更を加えます。

サポートされている地域

現在、X-Rayは、以下の AWS リージョン でVPC エンドポイントをサポートしています :

- 米国東部 (オハイオ)
- 米国東部 (バージニア北部)
- 米国西部 (北カリフォルニア)
- 米国西部 (オレゴン)
- アフリカ (ケープタウン)
- アジアパシフィック (香港)
- アジアパシフィック (ムンバイ)
- アジアパシフィック (大阪)
- アジアパシフィック (ソウル)
- アジアパシフィック (シンガポール)
- アジアパシフィック (シドニー)
- アジアパシフィック (東京)
- カナダ (中部)
- 欧州 (フランクフルト)
- ヨーロッパ (アイルランド)
- ヨーロッパ (ロンドン)
- ヨーロッパ (ミラノ)
- ヨーロッパ (パリ)
- ヨーロッパ (ストックホルム)
- 中東 (バーレーン)
- 南米 (サンパウロ)
- AWS GovCloud (米国東部)
- AWSGovCloud(米国西部)

のドキュメント履歴 AWS X-Ray

次の表は、 のドキュメントに対する重要な変更点を示しています AWS X-Ray。このドキュメントの更新に関する通知を受け取るには、RSS フィードにサブスクライブできます。

最新のドキュメント更新: 2023 年 2 月 8 日

変更	説明	日付
追加された機能	X-Ray は、、PutTraceSegments 、 GetTraceSummaries などのデータイベントBatchGetTraces をに記録するようになりましたAWS CloudTrail。また、X-Ray はGetSamplingStatisticSummaries 管理イベントを に記録するようになりました CloudTrail。詳細については、「 を使用した X-Ray API コールのログ記録 AWS CloudTrail」を参照してください。	2024 年 3 月 7 日
追加された機能	X-Ray は、W3C トレースIDsをサポートするようになりました。 OpenTelemetry W3C 詳細については、「X-Ray へのトレースデータの送信」を参照してください。	2023 年 10 月 25 日
追加された機能	Amazon SNS アクティブトレースを設定して、Amazon SNS トピックを通過するリクエストをトレースして分析できるようになりました。詳細については、Amazon SNS」	2023 年 2 月 8 日

	および AWS X-Ray「」を参照してください。	
X-Ray SDK for Node.js のトピックを更新	AWS SDK for JavaScript V3 を使用したクライアントの計測に関する詳細を追加しました。詳細については、「X-Ray SDK for Node.js を使用した AWS SDK 呼び出しのトレース」を参照してください。	2023 年 2 月 7 日
IAM 管理ポリシーの詳細を更新	`AWSXRayReadOnlyAccess` 、`AWSXRayFullAccess` 、`AWSXrayCrossAccountSharingConfiguration` の管理ポリシーにクロスアカウントオブザーバビリティ用の IAM アクセス許可を追加しました。詳細については、「X-Ray の IAM 管理ポリシー」を参照してください。	2023 年 2 月 7 日
追加された機能	AWS X-Ray でクロスアカウントオブザーバビリティ がサポートされるようになりました。これにより、 内の複数のアカウントにまたがるアプリケーションをモニタリングおよびトラブルシューティングできます AWS リージョン。詳細については、「クロスアカウントトレーシング」を参照してください。	2022 年 11 月 27 日

追加された機能	メッセージプロデューサー、Amazon SQS キュー、コンシューマー間のリンクされたトレースを表示できるようになりました。これにより、イベント駆動型アプリケーションから送信されたトレースを連結して表示できます。詳細については、「イベント駆動型アプリケーションのトレース」を参照してください。	2022 年 11 月 20 日
IAM 管理ポリシーの詳細を更新	リソースポリシーを一覧表示するための IAM アクセス許可を `AWSXRayReadOnlyAccess` 管理ポリシーに追加しました。詳細については、「X-Ray の IAM 管理ポリシー」を参照してください。	2022 年 11 月 15 日
IAM コンソールのアクセス許可と管理ポリシーの詳細を更新	X-Ray コンソールが使用する IAM アクセス許可のセットが、`AWSXRayReadOnlyAccess` 管理ポリシーの説明とともに更新されました。詳細については、「X-Ray コンソールの使用」を参照してください。	2022 年 11 月 11 日

AWS Distro for OpenTelemetry Ruby を追加	AWS Distro for OpenTelemetry (ADOT) には、分散トレースとメトリクスを収集するためのオープンソース APIsライブラリ、エージェントの単一のセットが用意されています。ADOT Ruby では、X-Ray やその他のトレースバックエンドの Ruby アプリケーションを計測できます。詳細については、AWS 「 Distro for OpenTelemetry Ruby」を参照してください。	2022 年 2 月 7 日
追加された機能	CloudWatch コンソールからトレースを表示し、X-Ray を設定できるようになりました。詳細については、「X-Ray コンソール」を参照してください。	2022 年 1 月 24 日
統合された CloudWatch RUM	AWS X-Ray と CloudWatch RUM を使用すると、ダウンストリームの AWS マネージドサービスを通じてアプリケーションのエンドユーザーから開始するリクエストパスを分析およびデバッグできます。詳細については、CloudWatch 「RUM」および AWS X-Ray「」を参照してください。	2021 年 12 月 3 日

の統合 AWS Distro OpenTelemetry	AWS Distro for OpenTelemetry (ADOT) には、分散トレースとメトリクスを収集するためのオープンソース APIsライブラリ、エージェントの単一のセットが用意されています。ADOT では、X-Ray やその他のトレースバックエンドのアプリケーションを計測できます。詳細については、「」を参照してください。アプリケーションの作成。	2021 年 9 月 23 日
追加された機能	AWS X-Ray が Amazon Virtual Private Cloud と統合されるようになりました。これにより、Amazon VPC 内のリソースは、パブリックインターネットを経由せずに X-Ray サービスと通信できるようになります。詳細については、「Using AWS X-Ray with VPC endpoints」を参照してください。	2021 年 5 月 20 日
追加された機能	AWS X-Ray が と統合されるようになり AWS CloudFormation、X-Ray リソースのプロビジョニングと設定が可能になりました。詳細については、「 を使用した X-Ray リソースの作成 CloudFormation」を参照してください。	2021 年 5 月 6 日

追加された機能	AWS X-Ray は Amazon と統合され EventBridge 、 を通過するイベントをトレースするようになりました EventBridge。これにより、ユーザーはシステムのより完全なビューを表示できます。詳細については、「Amazon EventBridge 」および AWS X-Ray「」を参照してください。	2021 年 3 月 2 日
ECR にデーモンを追加しました	デーモンは Amazon ECR からダウンロードできるようになりました。詳細については、「 daemon のダウンロード」ページを参照してください。	2021 年 3 月 1 日
追加された機能	AWS X-Ray は、Amazon へのインサイト関連の通知をサポートするようになりました EventBridge。これにより、 を使用してインサイトに対して自動アクションを実行できます EventBridge。詳細については、「X-Ray Insights を使用する」の「Insights 通知を有効にする」を参照してください。	2020 年 10 月 15 日
ダウンロード可能なデーモンを追加	AWS X-Ray では、Linux ARM64 のサポートデーモンが導入されました。詳細については、「」を参照してください。AWS X-Ray daemonBrazil WS	2020 年 10 月 1 日

追加された機能	AWS X-Ray は、Amazon CloudWatch Synthetics とのアクティブな統合をサポートするようになりました。これにより、応答時間やステータスなど、Synthetics canaryアクライアントノードの詳細を表示できます。Synthetics canaryアクライアントノードからの情報に基づいて Analytics コンソールで分析を実行することもできます。詳細については、CloudWatch「X-Ray を使用したSynthetics Canary のデバッグ」を参照してください。	2020 年 9 月 24 日
追加された機能	AWS X-Ray で の end-to-end ワークフローのトレースがサポートされるようになりました AWS Step Functions。ステートマシンのコンポーネントの視覚化、パフォーマンスのボトルネックの特定、およびエラーの原因となったリクエストのトラブルシューティングを行うことができます。詳細については、「AWS Step Functions 」および「 AWS X-Ray」を参照してください。	2020 年 9 月 14 日

追加された機能	AWS X-Ray では、アカウント内のトレースデータを継続的に分析し、アプリケーションの緊急の問題を特定するためのインサイトが導入されています。Insights はインシデントを記録し、解決するまでインシデントの影響を追跡します。詳細については、「X-Ray Insights の使用」を参照してください。	2020 年 9 月 3 日
追加された機能	AWS X-Ray では、Java 自動計測エージェントが導入されています。これにより、既存の Java ベースのアプリケーションを変更しなくてもトレースデータを収集できます。Java Web およびサーブレットベースのアプリケーションを、最小限の構成変更でコードの変更なしでトレースできるようになりました。詳細については、「」を参照してください。AWS X-Ray Java 用自動インストルメンテーションエージェント。	2020 年 9 月 3 日
追加された機能	AWS X-Ray は、トレースのグループの作成と管理を容易にするために、X-Ray コンソールに新しいグループページを追加しました。詳細については、「X-Ray コンソールでグループを設定する」を参照してください。	2020 年 8 月 24 日

追加された機能	AWS X-Ray では、グループとサンプリングルールにタグを追加できるようになりました。タグに基づいてグループおよびサンプリングルールへのアクセスを制御することもできます。詳細については、「」を参照してください。X-Ray のサンプリングルールとグループの作成そしてタグに基づいて X-Ray グループおよびサンプリングルールへのアクセスを管理する。	2020 年 8 月 24 日

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。